Beschreibungsbank zur rechnerunterstützten Verfolgung und Dokumentation großer Softwaresysteme

DISSERTATION
zur Erlangung des Grades eines
Doktor-Ingenieurs
der Abteilung für Elektrotechnik
an der Ruhr-Universität Bochum

von
Ulrich George
Altena, Westf.

Bochum 1982

Beschreibungsbank zur rechnerunterstützten Verfolgung und Dokumentation großer Softwaresysteme

DISSERTATION
zur Erlangung des Grades eines
Doktor-Ingenieurs
der Abteilung für Elektrotechnik
an der Ruhr-Universität Bochum

von
Ulrich George
Altena, Westf.

Bochum 1982

UNIV.
LEIDEN
BIBL.

Dissertation eingereicht am : . . . . . 5. Mai 1982

Tag der mündlichen Prüfung : . . . . . 26. Mai 1982

Hauptreferent : . . . . . Prof. Dr. W. Weber, Bochum

Korreferent : . . . . . Prof. Dr. K. Bender, Karlsruhe

INHALTSVERZEICHNIS

		Seite
1.	Die Softwarekrise	3
1.1	Die Diskrepanz Hardware-Software	3
1.2	Softwarequalität nimmt ab	3
1.3	Softwarekosten steigen rapide	5
1.4	Softwaretermine gleiten	8
1.5	Die Disziplin Software-Engineering	9
1.6	Ziel der Arbeit	11
2.	Prinzipien und Methoden des Software-Engineering	13
2.1	Die Phasen des Softwareprojekts	13
2.2	Die Analyse- und Spezifikationsphase	14
2.3	Die Planungsphase	15
2.4	Die Realisierungsphase	17
2.5	Die Testphase	21
3.	Das Software-Projektmanagement	23
3.1	Managementaspekte des Softwareproduktion	23
3.2	Das klassische Projektmanagement	24
3.3	Das Chief-Programmer-Team	26
4.	Engpässe der Softwareproduktion	29
4.1	Produktion und Kommunikation	29
4.2	Das Dokumentationsproblem	32
4.3	Die Teamorganisation	36
5.	Eine Beschreibungsbank für die Softwareproduktion	39
5.1	Eine Standortanalyse	39
5.1.1	Voraussetzung an die Systemumgebung	39
5.1.2	Die Akzeptanz durch die Benutzer	43
5.1.3	Minimale Implementierungsrechner-konfiguration	47

Seite

5.2 Die Daten der Beschreibungsbank 50

5.2.1 Klassifizierung der Informationsfülle 50

5.2.2 Erzeuger, Adressaten und Zugriffsberechtigungen 51

5.2.3 Hierarchiemodell und Programmierbereich 55

5.2.4 Die Beschreibung des Programmsystems 61

5.2.5 Die Beschreibung der einzelnen Module 64

5.2.5.1 Kenndaten allgemeiner Art 64

5.2.5.2 Beschreibung der Modulschnittstellen 66

5.2.5.3 Beschreibung des Modulablaufs 71

6. Die Realisierung der Beschreibungsbank 75

6.1 Die Umgebung der Beschreibungsbank SCROLL 75

6.2 Die Arbeit mit dem Beschreibungsbanksystem SCROLL 78

6.2.1 Die Dialogstufen 78

6.2.2 Aufgabenbereiche und Befehlstypen 82

6.2.3 Allgemeine Verwaltungsdienste 84

6.2.4 Aktualisierungs- und Protokolldienste 86

6.2.5 Auswertende Dienste 87

6.2.6 Generierende Dienste und Plotdienste 88

6.3 Der innere Aufbau der Beschreibungsbank 91

6.3.1 Allgemeine Gesichtspunkte zur Struktur 91

6.3.2 Datenhierarchie und Zugriffswege 93

7. Ausblick 97

Literatur 100

Zusammenfassung 108

Large System Programming has over the past decade been a tar pit.

F. P. Brooks jr.

1. Die Softwarekrise

Der provokative und kritische Titel des ersten Kapitels dieser Ausführungen soll auf ein Problem in der Datenverarbeitung aufmerksam machen, das in den letzten Jahren immer deutlicher wurde. Zunächst sollen anhand von Fakten die Symptome dieses Problems aufgezeigt werden.

1.1 Die Diskrepanz Hardware - Software

Es ist eine Tatsache, daß es heutzutage für fast jedes Problem, das mit Hilfe eines digitalen Rechners automatisiert werden kann, auf der Hardwareseite eine Schubladenlösung gibt. Für Automatisierungsvorhaben kleinerer Art steht auf dem Mikrorechnersektor ein verwirrendes Angebot von Rechnerbausteinen, deren Systemfamilien und - auf Standardkarten realisiert - ganzen Funktionsgruppen zur Verfügung, die - zumindest für den einzelnen Hersteller - weitestgehend miteinander kombinierbar sind und ein Baukastenangebot darstellen. Nach oben schließt sich lückenlos der Bereich der Minirechner an.
Diese Tatsache gilt jedoch nicht für die Software. Die unterschiedliche Art von Problemstellungen in verschiedenen Projekten bringt es mit sich, daß oft die gesamte Software neu zu erstellen ist. Speziell im Bereich der Prozeßdatenverarbeitung besteht relativ selten die Möglichkeit, existierende Softwarebausteine wie Real-Time-Betriebssystemkerne oder File-Manager ohne große Änderungen einzusetzen. Verschärfend zu diesem Problem kommt hinzu, daß die mittlere Größe von auf Prozeßrechnern implementierten Programmsystemen in den letzten Jahren ständig und rapide zugenommen hat. So wurde und wird der Bereich Software immer mehr zum Engpaß. Die Konsequenzen zeigen sich speziell in Qualität, Entwicklungskosten und Entwicklungszeitdauer.

1.2 Softwarequalität nimmt ab

Qualität ist ein Merkmal, das sich am schechtesten von allen hier aufzuzählenden Fakten in Zahlen ausdrücken läßt oder das greifbare Merkmale erlaubt, nach denen sie meßbar wäre.

Doch viel weniger trivial, als es zunächst erscheint, ist das einfache Merkmal, welche der geplanten Funktionen ein Programmsystem nach der Fertigstellung tatsächlich erfüllt. In der Tat enthalten zumindest die ersten Versionen aller Programmsysteme Fehler, zeigen also ein Verhalten, das der geplanten Arbeitsweise widerspricht.
Eine oft auftretende Folgeerscheinung für weitere Versionen wird von Brooks (/BRU 75/) als "Second System Effect" bezeichnet. In dem Bestreben, möglichst viele der alten Fehler zu beseitigen und möglichst viel nachzuholen, das die erste Version nicht bot, wird das Nachfolgesystem überladen und unrentabel.
Zwei Attribute zeichnen qualitativ gute Softwaresysteme aus:
die Benutzerfreundlichkeit und die Zuverlässigkeit.
Im heute zu beobachtenden Obergang von Stapelbetrieb zum Dialogbetrieb und dem Einsatz von Rechnern an Arbeitsplätzen, an denen immer mehr Datenverarbeitungslaien zu direktem Rechnerkontakt gezwungen sind, ist die Benutzerakzeptanz ein wichtiges Qualitätskriterium. Die folgenden Schlagworte spiegeln häufig genannte Kritikpunkte wieder:

- die Bedienung und Benutzung ist schwer erlernbar
- eine Falschbedienung führt zu ausweglosen Situationen
- eine Bedienungsunterstützung fehlt
- die Systemreaktionszeit ist zu lang.

Unter Berücksichtigung der Rechnereinsatzgebiete ist die Zuverlässigkeit wichtigstes Qualitätskriterium für Softwaresysteme. Hier soll der Begriff der Zuverlässigkeit weniger mit der exakten mathematischen Definition als vielmehr mit dem allgemeinen Sprachgebrauch verbunden werden. So ist - mit den Assoziationen Vertrauen oder Verlässlichkeit - Software dann zuverlässig, wenn für jede Eingabedatenkombination das vereinbarte Ergebnis erwartet werden kann.
Unzuverlässige Software kann unabsehbare Folgen mit sich bringen:

- es werden ohne Warnung unbeabsichtigt Benutzerdaten zerstört
- es entstehen hohe Ausfallkosten durch Produktionsstillstand
- es werden technische Anlagen oder im Extremfall Menschenleben in Gefahr gebracht.

Damit ist unzuverlässige Software wertlos, gleichgültig welche anderen positiven Eigenschaften sie aufweist.

In engem Zusammenhang zur Softwarequalität steht der Begriff der Wartung. Der Begriff Software-Wartung beinhaltet im allgemeinen Sprachgebrauch sowohl die Beseitigung auftretender Fehler wie auch die Umstellung,

Anpassung und Erweiterung von Softwaresystemen. Die erschreckend hohen Zahlen, die die Statistiken zur Wartung angeben - im Zusammenhang mit den Softwarekosten wird darauf näher eingegangen - zeigen deutlich, welchen qualitativ niedrigen Stand viele Softwaresysteme bei der Auslieferung haben.
Schließlich soll ein häufig zu beobachtender Effekt der Wartung nicht unerwähnt bleiben. Oft scheint die Beseitigung von Fehlern die Gesamtordnung eines Softwaresystems negativ zu beeinflussen, so daß im Extremfall die Behebung eines Fehlers unbeabsichtigt einen neuen Fehler hervorruft.

1.3 Softwarekosten steigen rapide

Aus einer Statistik (/POS 80/) sind folgende Daten zu entnehmen:
Ein typisches Prozeßrechnersystem verfügte 1975 über ca. 3K bytes Zentralspeicher zum Preis von 100 Dollar. Die Programmierung, vorwiegend in Assembler, dauerte etwa ein halbes Mannjahr und kostete 20 000 Dollar.
Im Jahr 1980 verfügte ein typisches Prozeßrechnersystem über einen Zentralspeicher von 40K bytes zum Preis von 100 Dollar, die Programmierung - vorwiegend in einer Hochsprache - dauerte etwa 1,5 Mannjahre und kostete 100 000 Dollar. Der hier verwendete Begriff des Mannjahrs hat sich als Maßeinheit für die Software-Erstellungsdauer eingebürgert. Ohne an diesem Begriff zunächst Kritik zu üben - dies wird später geschehen - zeigen diese nüchternen Zahlen folgende Tatsachen:

- Infolge des rapiden Preisverfalls durch Massenproduktion und Großintegration sind trotz wesentlich vergrößerter Leistungsfähigkeit die Hardwarekosten pro Funktion gesunken.
- Im gleichen Zeitraum sind die Softwarekosten um ein vielfaches gestiegen.

Softwarekosten sind zu 99 % Lohnkosten. So läßt sich die rapide Softwareverteuerung zunächst durch gestiegene Lohnkosten und die gleichzeitige, Hand in Hand mit der Hardware verlaufende Vergrößerung der Softwaresysteme erklären. Die Erfahrung zeigt, daß sich zusätzlich zu diesem sicher richtigen ein weiteres Faktum ergibt: Mit zunehmender Größe der Softwaresysteme steigen die Erstellungskosten nicht linear. Dieser Effekt wird später genauer zu untersuchen sein.
Bisher war ausschließlich von Softwareerstellungskosten die Rede. Es gibt jedoch Folgekosten zur Einsatzzeit des Softwaresystems: die Einsatzkosten und die Wartungskosten.

Direkte Einsatzkosten entstehen durch den fehlerfreien Einsatz und bestehen zum großen Teil aus Verbrauch von Betriebsmitteln wie Rechenzeit oder Druckerpapier. Sie sind prinzipiell nicht vermeidbar, ihre Verkleinerung ist direkt abhängig von der schon besprochenen Qualität des Softwaresystems.

Indirekte Einsatzkosten treten bei Fehlverhalten im Einsatz auf. In diese Rubrik fallen z.B. Ausfallkosten, Wiederholkosten oder Folgekosten nicht erkannter Fehler. Indirekte Einsatzkosten sind natürlich im Idealfall Null, sie sind stark abhängig vom Qualitätskriterium Zuverlässigkeit.

Folgekosten aufgetretener und bekannter Fehler sind die Wartungskosten, die bei der Beseitigung dieser Fehler und bei Modifizierung und Erweiterung der Softwaresysteme entstehen. Hier soll noch einmal auf die Statistik zurückgegriffen werden. Nach einem Forschungsergebnis der Infratest (/INF 76/) sind in der BRD 50% des Programmierpersonals ständig im Bereich der Wartung beschäftigt. Callender (/CAL 76/) nennt die Zahl von 200% der Entwicklungskosten, die zusätzlich zu Wartungsaufgaben für jedes Softwaresystem aufgebracht werden müssen. Boehm (/BOE 75/) ermittelte für bestimmte Software der amerikanischen Luftwaffe Entstehungskosten von 75 Dollar und Folgekosten für Wartung von bis zu 4000 Dollar pro Anweisung. Bild 1.3.1 zeigt die Kostenaufteilung bei einer typischen Prozeßautomatisierung durch einen digitalen Rechner, geschätzt bis ins Jahr 1985.

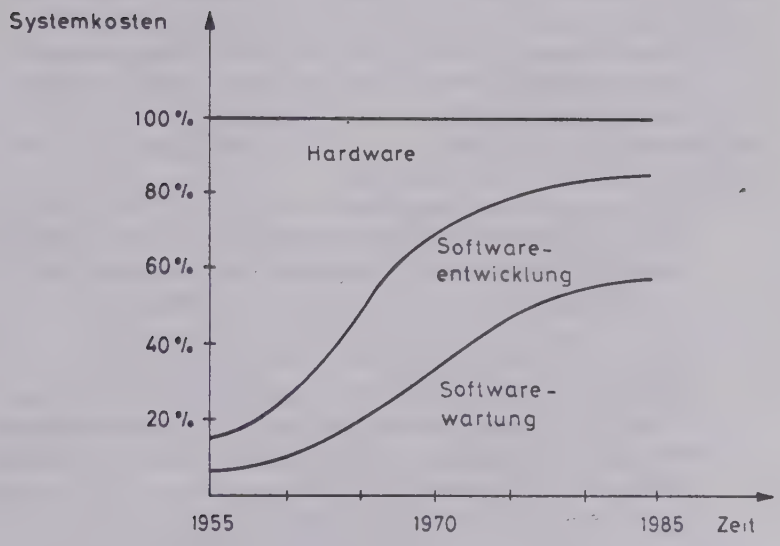


Bild 1.3.1 Kostenaufteilung zwischen Hardware znd Software bei typischer Prozeßrechnerentwicklung (nach /KOC 79/)

1.4 Softwaretermine gleiten

Ein letztes Mal werde die Statistik bemüht. Einer Analyse aus dem Jahre 1976 (/KOC 78/) zufolge benötigte man bei der Erstellung der Software für Prozeßautomatisierungen mit Digitalrechnern im Durchschnitt 53 % mehr Zeit als veranschlagt. Die Überschreitungen bei der Hardware fielen dagegen kaum ins Gewicht.
Damit werden im Bereich der Softwareerstellung wesentlich häufiger falsche Terminschätzungen gemacht als in anderen Ingenieurdisziplinen. Ein Grund könnte bei der starken Konkurrenz auf dem Softwaremarkt in der wissentlichen Angabe nicht haltbarer Termine zu sehen sein, mit dem Ziel, den Auftragszuschlag zu erhalten. Das eigentliche Problem scheint aber eher darin zu liegen, daß es bis heute keine realistischen Verfahren gibt, mit denen die Dauer eines Softwareprojektes berechnet werden kann. Vielmehr beruht die Terminaussage im allgemeinen auf der Schätzung eines erfahrenen Projektmanagers. Erschwerend kommt hinzu, daß bei der Softwareproduktion - im Gegensatz zur Hardware - weitgehend Unabhängigkeit von äußeren Einflüssen wie Lieferzeiten u.ä. besteht und die Terminschätzung somit eine Aussage der eigenen Fähigkeiten ist. Es scheint eine Psychologiefrage zu sein, daß gerade Schätzungen dieser Art zu Überschätzungen führen.
Bei Terminknappheit zeigt sich in der Praxis oft der Versuch, einen der folgenden beiden Auswege zu beschreiten:

- Zum laufenden Projekt werden weitere Mitarbeiter herangezogen. Dieses Verfahren führt jedoch höchstens zu geringen Terminverkürzungen. Der Regelfall ist jedoch sogar die Verschärfung des Terminproblems durch nachträgliche personelle Aufstockung.
  Dieser Effekt wird in den folgenden Kapiteln wiederzufinden sein.
- Die Einhaltung des Termins wird mit einer Verringerung des Funktionsumfangs erkauft. Es kommt dann häufig zur Auslieferung einer "vorläufigen Version", die die ihr zugedachte Aufgabe nur mangelhaft erfüllt. Die Gefahr des "Second System Effects" bei den folgenden Versionen wurde schon aufgezeigt.

Die drei Faktoren Qualität, Kosten und Zeit sind natürlich nicht voneinander unabhängig.

Bild 1.4.1 zeigt prinzipiell die Abhängigkeiten zwischen diesen Grundgrößen mit dem Kriterium "Entwicklungszeit" im Mittelpunkt.

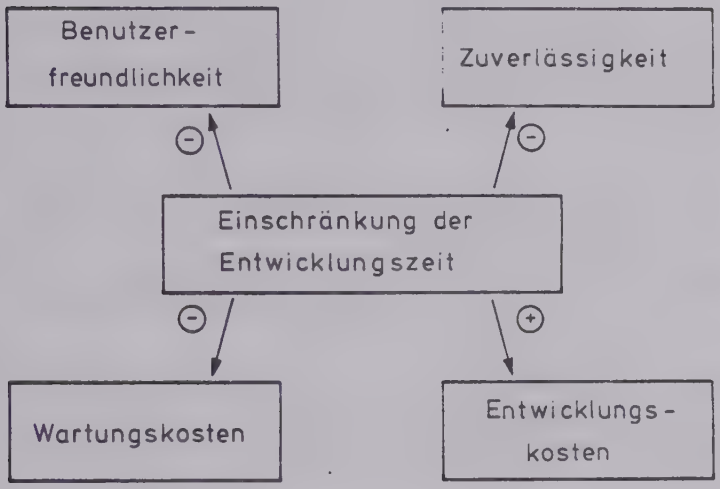


Bild 1.4.1: Positive (+) und negative (-) Einwirkungen auf die Grundgrößen der Softwareerstellung bei Einschränkung der Entwicklungszeit.

Die bei der Softwareerstellung wirtschaftlich relevanten Größen Zeit und Kosten stehen somit im Gegensatz zur Qualität und damit auch zu den anfallenden Folgekosten. Es besteht also die prinzipielle Notwendigkeit, bei der Erstellung jedes Softwareprodukts diese Faktoren gegeneinander abzuwiegen.

1.5 Die Disziplin "Software Engineering"

Die Zusammenfassung der bisherigen Kapitel ergibt folgende harte Aussage: Im Vergleich zur Hardware ist die Softwareerstellung weniger planbar, beherrschbar und kontrollierbar. Es gibt Unabwägbarkeiten in den Bereichen Aufwand, Kosten, Qualität, Akzeptanz, Wartung und vielen anderen mehr.

Die Softwareerstellung gleicht dem Sturz in eine Teergrube.

Schon Ende der sechziger Jahre erkannte man Teilaspekte der aufgezeigten Probleme, damals bei der Erstellung von Software für Großrechner wie dem Betriebssystem OS/360 für eine IBM-Rechnerserie.

Anfang der siebziger Jahre entstand der Begriff des "Software Engineering"; er sollte zeigen, daß man bemüht war, die Erstellung von Software zu einer eigenen Ingenieurdisziplin zu machen. Software sollte nicht länger intuitiv erzeugt werden, sondern ihre Erstellung sollte unter Einsatz ingenieurmäßiger Mittel beherrschbar werden.

Obwohl sich seitdem die Probleme verschärft haben, ist Software Engineering auch heute keine etablierte Ingenieurdisziplin. Den aktuellen Standpunkt spiegelt eher die provokativ getroffene Definition auf einer Software-Engineering-Tagung (/PRO 75/) wider:

"Software Engineering today is a collection of prescriptions, techniques and disciplines that show promise of bringing some order into the chaos that is computer programming."

Da jedoch der Begriff des Software Engineering inzwischen allgemein bekannt ist und auch Gegenstand der weiteren Ausführungen sein wird, soll er in der folgenden Definition /GEW 79/ festgelegt werden:

> Software Engineering ist die genaue Kenntnis und gezielte Anwendung von Prinzipien, Methoden und Werkzeugen für die Technik und das Management der Softwareentwicklung und -wartung auf Basis wissenschaftlicher Erkenntnisse und praktischer Erfahrungen, sowie unter Berücksichtigung des jeweiligen ökonomisch-technischen Zielsystems.

Der Begriff "Prinzip" in dieser Definition ist ein durch wissenschaftliche Erkenntnis oder durch Erfahrung gefestigter allgemeingültiger Grundsatz. Eine "Methode" ist eine einem Prinzip zugrundeliegende, zielgerichtete Vorgehensweise. Ein "Werkzeug" ist eine - meist mit Hilfe eines digitalen Rechners - automatisierte Methode.

In diesem Sinne beinhaltet diese Arbeit eine konsequent weitergedachte Methode des Software Engineering und ihre Unterstützung durch ein auf einem Mikrorechner realisierten Werkzeug: der Projektbeschreibungsbank.

1.6 Ziel der Arbeit

Das durch diese Arbeit angestrebte Ziel der Softwareproduktionsverbesserung bedingt zunächst eine genaue Betrachtung, Analyse und Kritik der aktuellen Softwareproduktion. Speziell wichtig erscheint die Erkennung von Engpässen als Ausgangspunkt für Überlegungen über deren Beseitigung.

Jedoch darf sich die Analyse der Softwareproduktion nicht nur auf den Vorgang der Produktion selbst beschränken, sondern es muß auch deren Umgebung betrachtet werden. Einer genaueren Untersuchung sind vor allem die Managementvorgänge zu unterziehen, die die Führung von Softwareprojekten zum Ziel haben.

So wird in den folgenden Abschnitten zunächst kurz das Phasenmodell vorgestellt, nach dem heute allgemein anerkannt der technische Ablauf eines Softwareprojekts zu erfolgen hat. Einer näheren Betrachtung unterliegen die Aspekte der Kommunikation und der Dokumentation, die sich in heutiger Zeit immer mehr als ungelöste Probleme erweisen.

Auf der Seite des Managementbereichs wird, ausgehend von dem von Baker und Mills vorgeschlagenen Chief-Programmer-Team eine Aktualitätsanalyse dieser Struktur und der damit verbundenen Personen und Aufgabenbereiche durchgeführt, deren Ergebnis vor allem in einer Änderungsforderung im Aufgabenbereich des Projektsekretärs besteht.

Die eigentliche Grundidee der durchgeführten Arbeit besteht in der Unterstützung verschiedener Aspekte der Softwareproduktion durch den Rechner selbst. Datenbankähnlich werden alle allgemeinen, managementbezogenen und fachlichen Daten des laufenden Softwareprojekts durch den Rechner gespeichert und im Dialog mit den einzelnen Programmierern verwaltet und bearbeitet.

Diese Idee ist in ihren Grundsätzen nicht neu. Es gibt bereits eine Reihe rechnerunterstützter Hilfsmittel zur Softwareprojektunterstützung, genannt seien als Auszug z.B. EPOS (/BIE 79/), PET/X1150 (/SOF 81/), MAPSOFT (/DAN 80/) oder DARTS (/GEI 80/).

Deshalb sollen im folgenden die wichtigsten Eigenschaften des im Rahmen dieser Arbeit entstandenen Systems SCROLL genannt werden, die in ihrer Zusammenfassung dies neu entstandene System abgrenzen und hervorheben.

- SCROLL ist auf einem Mikrorechner implementiert.
  Im Gegensatz zu den meisten bekannten Systemen, die Großrechnereinsatz oder die Existenz eines speziellen Minirechners voraussetzen, werden die Investitionskosten beim System SCROLL klein gehalten. Bei einer geschätzten Preisklasse von 20000...40000 DM für die Hardware ist sein Einsatz attraktiv, zumal auf dem gleichen Rechner auch Programmentwicklung betrieben werden kann. Auch die alleinige Nutzung des Rechners zu Projektverfolgungszwecken ist möglich.

- SCROLL arbeitet vollständig dialogorientiert.
  Nachdem in anderem Zusammenhang der Begriff und die Hintergründe der Benutzerfreundlichkeit schon besprochen wurden, ist es selbstverständlich, diese Gedanken in ein projektbegleitendes Softwaresystem einfließen zu lassen. Das System SCROLL arbeitet vollständig am Terminal im Dialog mit dem Benutzer; der Dialog basiert auf einer rechnergeführten Menü-Technik.

- SCROLL unterstützt Prozeßdatenverarbeitungs-Projekte.
  Softwaresysteme im Mikrorechnerbereich sind in der Regel Systeme mit Echtzeitverhalten. SCROLL unterstützt diese wesentlich von sequentiell ablaufenden Systemen abweichenden Eigenschaften durch spezielle Funktionen zur Bearbeitung paralleler Prozesse.

- SCROLL beinhaltet Datenverwaltung, Datenauswertung und Datengenerierung.
  Im System SCROLL werden die projektbegleitenden und dokumentierenden Daten nicht nur datenbankähnlich verwaltet, sondern auch nach bestimmten Kriterien überprüft und ausgewertet. Weiterhin sind neue Daten automatisch erzeugbar, die im Projekt einsetzbar sind und damit die Arbeit der Programmierer unterstützen.

Divide et impera

- teile und herrsche

römisches Sprichwort

2. Prinzipien und Methoden des Software Engineering

Als Grundlage und zum Verständnis der folgenden Ausführungen sollen in diesem Abschnitt die Prinzipien und die wichtigsten anerkannten Methoden der Disziplin Software Engineering zusammengefaßt werden.

2.1 Die Phasen des Softwareprojektes

Ausgangspunkt der folgenden Betrachtungen sind die inzwischen allgemein anerkannten Phasen, in denen ein Softwareprodukt entstehen sollte. Man findet in der Literatur diese Phasen in verschiedener Aufteilung und mit verschiedener Nomenklatur, inhaltlich jedoch stimmt die Vorgehensweise größtenteils überein. Die hier gewählte Phasenaufteilung stützt sich auf das folgende Phasenmodell ab:

- Analyse und Spezifikation
  Es wird eine exakte Definition der durch ein Softwaresystem zu lösenden Aufgabe erarbeitet. Die Schnittstellen zum Benutzer und zur Hardware werden detailliert beschrieben.
- Planung
  Die nun vollständig bekannte Aufgabe wird in einzeln zu realisierende und überschaubare Teilaufgaben aufgespalten und deren Schnittstellen festgelegt.
- Realisierung
  Jede Teilaufgabe wird in ihrem logischen Aufbau und Ablauf festgelegt und in der gewählten Programmiersprache codiert.
- Test
  Jede realisierte Teilaufgabe wird einzeln auf das Auftreten logischer Fehler überprüft.

Bild 2.1.1 zeigt eine typische Aufteilung dieser Phasen auf die gesamte Systemerstellungszeit.

Jede der aufgezeigten Phasen soll im folgenden einzeln besprochen werden. Dabei wird auf wichtige Methoden eingegangen und Werkzeuge vorgestellt.

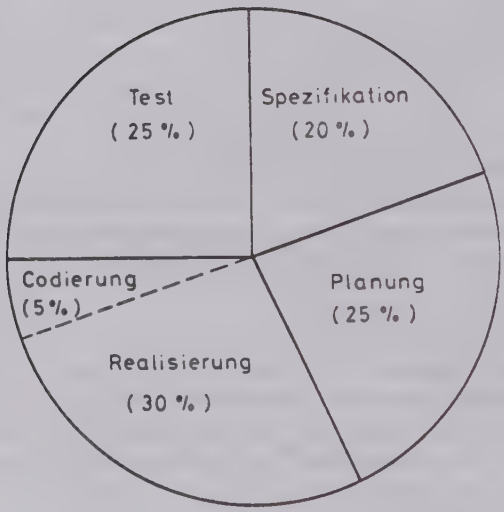


Bild 2.1.1 Prozentuale Aufteilung der Softwaresystemerstellungszeit in Produktionsphasen

2.2 Die Analyse- und Spezifikationsphase

Bei dem Versuch, eine bestimmte Aufgabe mit Hilfe eines digitalen Rechners und eines darauf implementierten Softwaresystems zu lösen, treffen in der Regel Personen aufeinander, die eine vollständig voneinander abweichende Fachsprache sprechen. Auf der einen Seite stehen der Auftraggeber und die späteren Systembenutzer, die ihr Problem kennen, im allgemeinen aber auf dem Gebiet der Datenverarbeitung keine Kenntnisse besitzen und auch nicht besitzen müssen. Auf der anderen Seite hat das Programmiererteam detaillierte Kenntnisse über Aufbau und Programmierung digitaler Rechenautomaten; jedoch ist zumindest ein Teil dieser Gruppe gezwungen, sich zusätzlich Kenntnisse über das geplante Einsatzarbeitsgebiet anzueignen.

Die genaue Analyse des Einsatzgebietes und seiner Umgebung ist deshalb der Ansatzpunkt beim Start eines Softwareprojektes. Es wird ein Benutzermodell spezifiziert, das möglichst exakt folgende Daten beinhalten sollte:

- Wissensstand der späteren Benutzer und ihre Wünsche und Anforderungen an das neue System
- zu beachtende Vorschriften und Einschränkungen für das neue System
- die Gesamtheit aller möglichen einzugebenden Daten und Anweisungen und die darauf erwarteten Ausgabedaten und Meldungen
- die für den Einsatz geplante Hardwarekonfiguration, die Systemsoftware und die zur Verfügung stehenden Entwicklungshilfen.

Man bezeichnet die Zusammenfassung dieser Daten in schriftlicher Form oft als "Pflichtenheft". Das Pflichtenheft dient als Grundlage der Verständigung zwischen Systemersteller und Systembenutzer und ist meist Grundlage des Vertrags zwischen diesen Partnern.

2.3 Die Planungsphase

Nach der Analyse und Spezifikation des zu erstellenden Softwareprodukts kann nun die Synthese beginnen, die Erstellung der Software wird in Angriff genommen. Ausgangspunkt der sich anschließenden Planungsphase ist folgende Situation:
Es existiert eine (reale) Basismaschine, die Hardware des einzusetzenden Rechnersystems. Diese Maschine ist in ihren möglichen Eingaben, ihren Reaktionen und ihren Ausgaben durch Beschreibungen des Herstellers genau definiert. Auf der anderen Seite ist das zu erstellende Softwareprodukt mit seinen Eingaben, seinen Reaktionen und Ausgaben durch die Spezifikation genau festgelegt. Diese sogenannte Benutzermaschine ist abstrakter oder virtueller Natur, sie existiert durch Abbildung mit Hilfe von Software auf der Basismaschine.

Ziel der Planungsphase ist diesem Modell folgend die Zusammenführung von Benutzer- und Basismaschine. Dies geschieht durch Einfügung weiterer Abstraktionsebenen,also durch die Definition weiterer virtueller Maschinen bestimmter Detaillierung und bestimmter Fähigkeiten.

Lange Zeit wurde diese Zusammenführung intuitiv von unten nach oben ("bottom-up") vorgenommen. Einzelne Programmierer bauten ohne große Planung, ausgehend von der gegebenen Maschinensprache oder vom Anweisungsvorrat einer höheren Programmiersprache komplexere Programmteile auf, die schließlich zusammengefügt und - meist mit Mühe - zum Laufen gebracht wurden.

Schnupp und Floyd /SCH76/ vergleichen dieses Vorgehen sehr anschaulich mit dem Dreizehnjährigen, der mit Hilfe seines Metallbaukastens einen Kran aufbauen will. In dem Bestreben, möglichst schnell ein fertiges Teilprodukt zu erstellen, beginnt er mit dem Kranhaken, baut daran den Ausleger und stellt beim Anbau an den Rumpf die ersten Konstruktionsfehler fest, die ihn zum Umbau und zur Improvisation zwingen.

Bottom-up-entwickelte Softwaresysteme sind im allgemeinen unsystematisch aufgebaut, sie enthalten unklare Schnittstellen und sind somit unübersichtlich. Auftretende Fehler führen zu Schwierigkeiten bei der Wartung, geplante Erweiterungen stoßen auf Probleme, da keine klaren Ansatzpunkte erkennbar sind.
Dijkstra /DAH72/ postulierte zuerst eine in anderen Ingenieurdisziplinen schon lange anerkannte Vorgehensweise: Ausgehend von der Aufgabe werden sukzessive so lange Teilaufgaben gebildet, bis Funktionen vorliegen, die sich leicht in ein Programm umsetzen lassen. Ziel dieser Vorgehensweise von oben nach unten ("top-down") ist der Entwurf einer baumartigen, hierarchischen Programmstruktur. Wurzel dieses Baums ist - mit der größten Abstraktion - die Benutzermaschine; von oben nach unten wird jede Aufgabe jeweils in Teilaufgaben mit größerem Detaillierungsgrad aufgespalten.

Der zu schließende Raum zwischen der Benutzermaschine und der Basismaschine wird somit in Ebenen aufgeteilt. Jede Ebene beinhaltet eine Anzahl virtueller Maschinen mit definierten Eingaben, einer Menge grundlegender Operationen und definierten Ausgaben. Damit wird das System aufgespalten in Teilsysteme und Untersysteme von Teilsystemen mit dem Ziel, Komplexität zu reduzieren und schließlich eine Abbildung auf der existierenden Maschine zu erreichen.Folgende Regeln - in kurzer Zusammenfassung - gelten für die Aufspaltung:

- Ausgangspunkt der Aufspaltung ist ein einzelner Knoten, die Benutzermaschine.
- das aufgespaltene System soll hierarchisch und baumartig strukturiert sein, insbesondere sind Graphen auf die gleiche Ebene, über mehrere Ebenen hinweg und Schleifenbildung verboten.
- jede Teilaufgabe soll in ihren Ein- und Ausgaben, nicht jedoch in ihrem internen Aufbau ("information hiding", /PAR72/) der sie benutzenden Aufgabe eine Ebene darüber bekannt sein.

- die Teilaufgaben (in diesem Zusammenhang oft auch "Module" genannt) sollen unabhängig von verschiedenen Programmierern realisiert und getestet werden können.

Das ist viel und doch wenig, denn diese Regeln geben zwar Richtlinien über den zu erstellenden Aufbau, geben aber keine Kriterien für das Vorgehen beim Aufspalten einer konkreten Aufgabe. Vorgeschlagene Verfahren für dieses Gebiet sind bis heute umstritten und ein Durchgreifen auf die Praxis nur in Teilgebieten erkennbar. Methoden wie die HIPO-Technik /IBM75/, Jackson-Methode /JAC75/, SADT-Methode /ROS77/ oder LITOS /SCHU78/ wurden in Einzelprojekten benutzt, haben sich aber allgemein nicht durchsetzen können.
So wird - durchaus unter Kenntnis und Beachtung dieser Methoden - die Planung bis auf weiteres vor allem der Intuition eines erfahrenen Systementwicklers unterliegen.

2.4 Die Realisierungsphase

Nach erfolgtem Entwurf des Programmsystems als eine Menge einzeln bearbeitbarer Teilaufgaben bezeichnet man als Realisierung alle Tätigkeiten, die notwendig sind, um diese Teilaufgaben in ein ablauffähiges Programm umzusetzen. An dieser Stelle sind alle grundsätzlichen Entscheidungen über die Struktur des Softwaresystems getroffen, das System ist in seinem Aufbau festgelegt. Umgekehrt machen sich Entscheidungsfehler vor diesem Zeitpunkt bei ihrer späteren Aufdeckung besonders gravierend bemerkbar, denn oft wird dann ein großer Teil der folgenden Arbeit vernichtet.

Einen besonderen Hinweis verdient der Effekt, daß zu Beginn der Realisierungsphase etwa 50 % der gesamten Entwicklungszeit verstrichen sind, ohne daß eine einzige Programmzeile zustande kam, eine Tatsache, die sicher schon manchen nicht mit der Materie vertrauten Manager einige graue Haare gekostet hat. Aber - Bild 2.4.1 zeigt es deutlich - den Fortgang des Projektes zu diesem Zeitpunkt an fertigen Teilprogrammen zu messen ist grundfalsch; vielmehr ist eine exakte Planung Grundlage des Projekterfolgs.

Aufgabe der einzelnen Programmierer ist nun die Aufspaltung der ihnen zugewiesenen Teilaufgaben in Module und die Entwicklung eines Lösungsalgorithmus, also die Festlegung des logischen Ablaufs in jedem Modul. Damit ist das Modul die kleinste Einheit der Aufgabenunterteilung und berücksichtigt erstmals im Projektverlauf die Realisierung in einer gewählten Programmier-

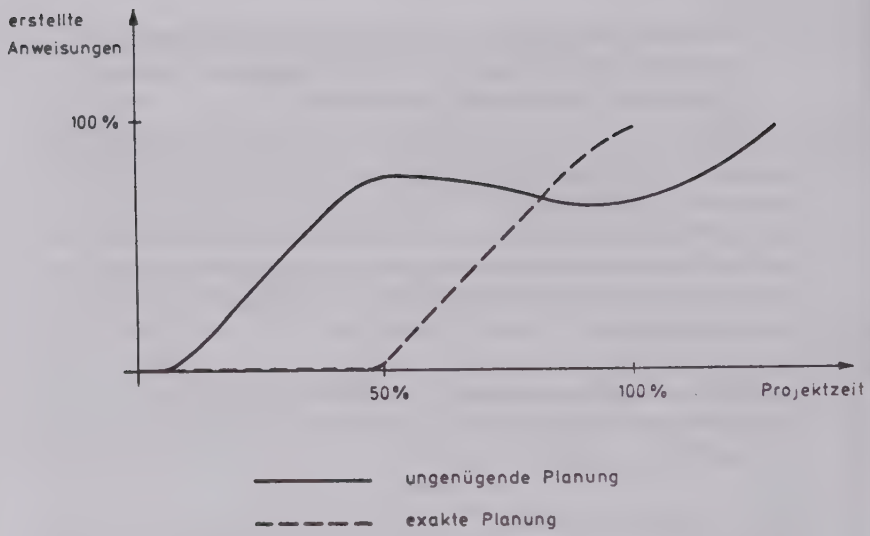


Bild 2.4.1 nach /ODG72/
Projekterstellungszeit in Abhängigkeit der Planungsphase

sprache, indem es folgende Eigenschaften besitzen sollte:

- ein Modul beinhaltet einen vollständigen Algorithmus oder einen logisch in sich geschlossenen Teil eines Algorithmus
- die Anweisungen eines Moduls erscheinen physikalisch zusammen und werden zusammen übersetzt
- die Anweisungen eines Moduls werden durch Modul-Klammer-Anweisungen (z.B. DO und END) begrenzt und sind über den Modulnamen gemeinsam identifizierbar.

In der Realisierungsphase existieren Methoden, die den Programmierer in dem Bemühen unterstützen sollen, eine geeignete logische Struktur in den Ablauf der Einzelmodule zu bringen.

Eine mathematisch besonders gut fundierte Methode ist die Entscheidungstabellen-Technik /ELB73/. Ausgangspunkt bei deren Anwendung ist die formale Aufstellung einer Tabelle, die alle logisch möglichen Bedingungen und die Gesamtheit aller vorgesehenen Aktionen der zu lösenden Aufgabe

enthält. Der vollständige Satz der im zweiten Schritt aufzustellenden Entscheidungsregeln enthält alle Kombinationen von Aktionen, die auf bestimmte Bedingunskombinationen zu erfolgen haben. Der formale Aufbau der Tabellen ermöglicht dem Programmierer - auch rechnerunterstützt (Entscheidungstabellen-Prozessoren) - eine Oberprüfung auf Vollständigkeit, Redundanzfreiheit und Widersprüche. Daß sich diese Methode trotz der genannten Vorteile nicht durchsetzen konnte, liegt wohl an deren formalem Charakter, der eine ebenso mathematisch formale Denkweise des Programmierers voraussetzt.

Genau aus diesem Grund hat die den intuitiven Programmierstil unterstützende Methode der Datenflußpläne und der Programmablaufpläne ausgesprochen weite Verbreitung gefunden. Datenflußpläne sind durch Graphen verbundene Symbole, die die Bearbeitung der Daten in einem System verfolgen, Programmablaufpläne zeigen anhand durch Graphen verbundener Symbole den logischen Ablauf der Operationen in einem System. Die zu verwendenden Symbole sind inzwischen in DIN 6001 /DIN 75/ genormt. Jedoch birgt die allzu sorglose Verwendung dieser Methode die Gefahr, daß der interne Aufbau der Einzelmodule vollkommen unübersichtlich wird (Fachjargon:"Spaghettiprogramm"), ein Problem, das man bei der vorherigen Phase ja gerade durch eine exakte, hierarchische Planung zu vermeiden suchte.
Eine gute, aber leider noch zu wenig bekannte Alternative sind die von Nassi und Shneiderman /NAS73/ vorgeschlagenen Diagramme. Nach dieser Technik werden einige wenige Symbole nach einem im folgenden näher erläuterten Konzept ineinandergeschachtelt und aneinandergereiht. Bild 2.4.2 zeigt zunächst die möglichen Grundelemente dieser Technik und ihr Analogon als Programmablaufplan. Im Gegensatz zum Programmablaufplan, der die Verbindung beliebiger Symbole durch Graphen erlaubt, ist die Verwendung der Nassi/Shneiderman-Elementarsymbole wie folgt reglementiert:

- die Elementarsymbole dürfen beliebig aneinandergereiht werden
- jedes Elementarsymbol 1 (Operation), das als Teilsymbol der Elementarsymbole 2 bis 5 vorkommt, darf durch eine beliebige Aneinanderreihung der Elementarsymbole 1 bis 5 ersetzt werden. Damit wird eine Schachtelung realisiert, die einer Technik der schrittweisen Verfeinerung entspricht.

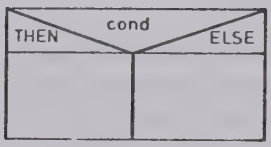
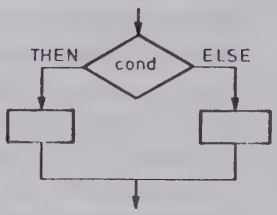
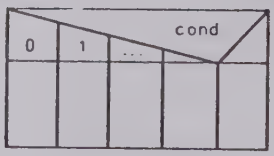
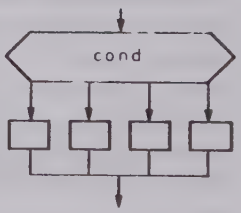
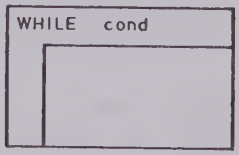
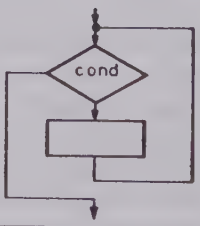
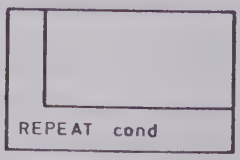
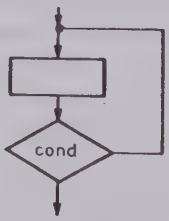
	Nassi / Shneiderman	Programm-ablaufplan
1 Operation		
2 Verzweigung (condition)		
3 Mehrfach-verzweigung (switch)		
4 pre-checked loop		
5 post-checked loop		

Bild 2.4.2 Grundsymbole der Methode nach Nassi/Shneiderman und die entsprechenden Elemente der Programmablaufplanmethode

Bei zeichnerischer Verwendung der Elementarsymbole sind durch deren physikalische Anordnung die genannten Regeln immer und automatisch erfüllt.
Nach dieser Methode aufgebaute Algorithmen zeigen deren logische Struktur, sie sind streng strukturiert, damit übersichtlich und leicht wartbar, denn

- der dynamische Ablauf erfolgt streng von oben nach unten, Rücksprünge sind nur kontrolliert in Schleifen möglich
- Elementarsymbole liegen vollständig ineinander oder sind gereiht, es gibt keine teilweisen Oberschneidungen
- jedes Elementarsymbol in jeder Schachtelungstiefe hat genau einen Eingang und genau einen Ausgang.

Alle Realisierungsmethoden sollten bis zum vollständigen logischen Entwurf der Module programmiersprachenunabhängig verwendet werden. Dies beinhaltet insbesondere, daß die Operationen in Programmablaufplänen oder Nassi/Shneiderman-Diagrammen nicht als Folge von Statements einer Programmiersprache erscheinen, sondern verbalen umgangssprachlichen Text beinhalten.

Die Umsetzung der entworfenen Algorithmen in syntaxfehlerfreien Code einer gewählten Programmiersprache ist der letzte Schritt der Realisierung. Bei richtiger Vorbereitung besteht diese Codierung in weiten Teilen aus mechanischem Umsetzen von Operationen und Kontrollstrukturen. Speziell die Elementarsymbole der Nassi/Shneiderman-Methode werden durch die meisten Programmiersprachen unterstützt.
Die eigentliche Codierung, also die Formulierung der logischen Struktur in einer Programmiersprache, die Eingabe der Quellen in den Rechner und die Korrektur der Schreibfehler bis zur syntaxfehlerfreien Obersetzung wird bei einer derart guten Vorbereitung mit nur 5 % der Systemerstellungszeit veranschlagt. Erst nach über 60 % der geplanten Systemerstellungszeit findet also der erste Rechnerkontakt statt, und es entstehen die ersten "lines of code".

2.5 Die Testphase

Der Test der erstellten Einzelmodule und ihr Zusammenwirken als Gesamtsystem wird von einigen Autoren als letzter Teil der Realisierung betrachtet. Wegen seiner zentralen Bedeutung und seiner Länge von bis zu 25 % der Systemerstellungszeit soll er jedoch hier als eigene Phase betrachtet werden.

Bei einer vollkommen exakten Planung und Realisierung wäre theoretisch kein Test nötig. Unter dieser Prämisse ist der Test als eine Folge der menschlichen Unzulänglichkeit zu sehen. Der semantischen Fehlermöglichkeiten gibt es viele: angefangen bei der Planungsphase durch Wahl einer falschen Aufteilung oder der falschen Absprache von Schnittstellen über den vergessenen Zweig bei der Realisierung einer logischen Struktur bis hin zur fehlenden Vorbesetzung einer Variablen gibt es eine Unzahl von Möglichkeiten.

Test ist unproduktiv. Er erzeugt keine positiven Ergebnisse, sondern allenfalls negative: er deckt Fehler auf, oder anderenfalls führt er zu der Frustration völlig nutzloser Betätigung.
Die Beendigung der Testphase ist willkürlich. Denn auch ein noch so ausführlicher Test kann zwar bei genügender Suche zur Entdeckung eines weiteren Fehlers führen, nie aber die Aussage der Fehlerfreiheit erzwingen.
So ist es Brauch, daß sich Softwaresysteme nach der Auslieferung "bewähren" müssen und der Ersteller sich bereiterklärt, alle in den ersten Einsatzmonaten auftretenden Fehler kostenlos zu beseitigen. Es ist nur zu verständlich, daß das Chaos, das ungenügend getestete Software beim ersten Anwendereinsatz verursacht, nicht gerade zur Akzeptanz und zum Vertrauen in das neue Produkt beiträgt.

Trotzdem sollte das Testen auf ein - allerdings effektives - Minimum beschränkt werden. Das geschieht zum einen schon bei der Planung und Realisierung durch sorgfältiges Arbeiten und der Erstellung einer Teststrategie und eines Testrahmens, zum anderen sollte der Test selbst gezielt durchgeführt werden, um mit höchster Wahrscheinlichkeit die existierenden Fehler zu finden.

Die Erfahrung zeigt, daß viele Fehler trotz allen Aufwands erst nach Jahren des Einsatzes auftreten und dann,wie beschrieben,zu besonders hohen Folgekosten durch Wartung führen.

"Would you tell me, please,
which way I ought to go from here ?" said Alice.
"That depends a good deal on where you want to get to ", said the cat.
"I don't know where ", said Alice.
"Then it doesn't matter which way you go ", said the cat.

L. Caroll: Alice in Wonderland

3. Das Software-Projektmanagement

Eine Beschreibung und Analyse des Softwareentwicklungsprozesses wäre unvollständig ohne einen - wenn auch kurzen - Abschnitt über die organisatorischen Probleme: das Management.

3.1 Managementaspekte der Softwareproduktion

Die Probleme der Softwareerstellung wären grundlegend einfacher, wenn die alleinige Aufgabe darin bestände, zu einem beliebigen Zeitpunkt ein funktionsfähiges Produkt zu liefern. Aber natürlich spielen die wirtschaftlichen Aspekte eine entscheidende Rolle. Teil des Softwareproduktionsvertrags ist sicherlich ein zugesagter Fertigstellungstermin, und auch über den Aufwand, also die entstehenden Kosten, hat man sich im allgemeinen vor Produktionsbeginn geeinigt.
So besteht die Notwendigkeit, jede Softwareentwicklung, die von mehreren Personen in Teamarbeit durchgeführt wird, zu überwachen und zur richtigen Zeit die richtigen zum Erreichen des Projektziels nötigen Entscheidungen zu treffen.
Diese Oberwachung ist in zwei zunächst voneinander unabhängig erscheinenden Bereichen durchzuführen:

- Die fachliche Oberwachung hat das Ziel, die Qualität des entstehenden Produkts zu garantieren.
- Die wirtschaftliche Oberwachung soll für die Einhaltung der Termin- und Aufwandsvereinbarungen sorgen.

Die fachliche Oberwachung allein ist sicherlich die einfachere; es gibt konkret spezifizierte und überprüfbare Funktionen, die das System letztendlich erfüllen muß.

Die schon ausführlich beschriebenen Probleme entstehen mit den wirtschaftlichen Vorgaben und der wirtschaftlichen Oberwachung. Drei Ursachen kennzeichnen deren Auftreten.
Zunächst ist es - wie bereits beschrieben - bis heute schwer, exakte Aussagen über die Größe eines geplanten Softwareprojektes zu machen. Eine extrem falsche Einschätzung dieser Grundgröße führt jedoch im Ansatz zu wirtschaftlichen Vorgaben, die nicht haltbar sind.
Weiterhin ist der Teamleiter eines Softwareprojektes als Informatiker oft wesentlich besser in der Lage, eine fachliche Oberwachung als eine wirtschaftliche Oberwachung durchzuführen. Tatsächlich verlangen die beiden Teilaufgaben völlig unterschiedliche Qualifikationen, die in einer Person vereint sein müssen.
Schließlich kommt hinzu, daß die voraussichtliche Einhaltung wirtschaftlicher Vorgaben während des Projektverlaufs wesentlich schwerer überprüfbar ist. Aussagen wie: "90 % codiert, 50 % getestet " erweisen sich als nicht genügend aussagekräftig. So fehlen bei ungenügender Struktur einfach eindeutig erkennbare Zwischenergebnisse, auf deren Erreichen sich die wirtschaftliche Oberwachung abstützen könnte.

3.2 Das klassische Projektmanagement

Ein Modell der in den 60er-Jahren entstandenen und auch heute noch bei vielen Softwareprojekten etablierten Managementstruktur zeigt Bild 3.2.1.

Zentrale Figur dieses Modelles ist der Projektleiter. Er ist zuständig für die fachliche und wirtschaftliche Oberwachung des Programmiererteams und bekommt von den einzelnen Programmierern Erfolgsmeldungen über die Fertigstellung der ihnen zugeteilten Teilaufgaben. Er berichtet, falls dieser daran interessiert ist, dem Auftraggeber über den technischen Fortschritt und die technische Realisierung des Projektes.
Die wirtschaftliche Verantwortung des Projektes liegt bei einem Systemmanager, der im allgemeinen mehrere Projekte gleichzeitig betreut. Er erhält vom Projektleiter die wirtschaftlich interessanten Neuheiten und gibt diese aufbereitet an den Auftraggeber weiter. Während er in wirtschaftlichen Belangen gegenüber dem Projektleiter weisungsbefugt ist, hat er fachlich keinen Einfluß auf den Ablauf.

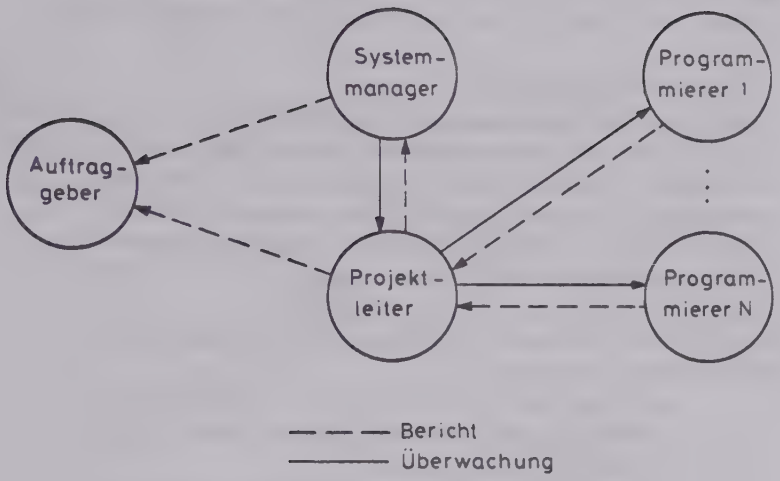


Bild 3.2.1 Standardstruktur des Software-Projektmanagements

Die Programmierer schließlich führen selbständig den Entwurf und die Realisierung der ihnen zugewiesenen Programmteile durch. Die Absprache von Schnittstellen erfolgt möglichst unter Hinzuziehung des Projektleiters. Erst beim Integrationstest werden alle fertigen Programmteile zusammengefügt und auf ihr Zusammenwirken geprüft.

Bei kleineren Projekten kommt es oft zu Abweichungen von diesem Modell. So kann z.B. der Projektleiter selbst aktiv als Programmierer an der Erstellung eines Projektteils arbeiten, oder der Projektleiter ist sein eigener Systemmanager.

Mit dem Erkennen der Softwarekrise begannen Zweifel an diesem Managementmodell. Ansatz der Kritik war die personelle Trennung der wirtschaftlichen und fachlichen Oberwachung. Der technisch wenig versierte Manager ist einfach nicht in der Lage, aus den Berichten des Projektleiters verläßlich die Situation des Projektes zu beurteilen.

Mit einem Softwareprojekt für eine New Yorker Zeitung - eins der ersten Projekte, die streng nach dem top-down-design durchgeführt wurden - führte die Fa. IBM das erste Mal eine neue Managementstruktur ein, die dann von Baker und Mills /BAK73/ vorgestellt wurde.

3.3 Das Chief-Programmer-Team

Grundsätzliche Idee des Managementvorschlages von Baker und Mills war die Einsicht, daß exakt definierbare Ereignisse spezifiziert werden müßten, deren überprüfbares Eintreten zur Kontrolle des Projektfortschritts verwendbar sei. Diese Ereignisse müssen über den Projektablauf etwa gleichverteilt sein. Der fachliche Hintergrund, der mit dem Auftreten eines Ereignisses verbunden ist, soll für die wirtschaftliche Oberwachung uninteressant sein.
Baker und Mills nennen zwei Voraussetzungen für die Anwendung ihrer Chief-Programmer-Team-Struktur:

- das Projekt muß streng nach den Phasen des Software-Engineering unter Einsatz der top-down-Strukturierung durchgeführt werden.
- es muß eine Projektbibliothek eingerichtet werden, in der alle projektrelevanten Daten wirtschaftlicher und fachlicher Art gesammelt werden.

Es gibt in der neuen Organisationsform drei fest definierte Figuren mit genau spezifizierbaren Aufgaben.

Der Projektleiter (Chief Programmer) ist verantwortlich für den fachlichen Ablauf des Projektes. Er erstellt in Zusammenarbeit mit dem Auftraggeber die Spezifikation, und er führt die Projektplanung durch, teilt also das Projekt in hierarchisch aufeinander aufbauende Teilaufgaben mit definierten Schnittstellen auf. In der Realisierungsphase setzt er Programmierer ein, die er laufend mit der Erstellung dieser einzelnen, jetzt genau definierten kleinen Teilaufgaben beauftragt und deren Abnahme er überwacht.
Der Projektassistent (Backup Programmer) ist als engster Mitarbeiter des Projektleiters sein Diskussionspartner und Stellvertreter. Seine fachliche Qualifikation muß ausreichen, um den Projektleiter bei der Spezifikationserstellung und der Planung zu unterstützen. Er ist der einzige neben dem Projektleiter, der das gesamte Projekt fachlich überblickt und der somit bei kurzfristigem Ausfall des Projektleiters (Krankheit,

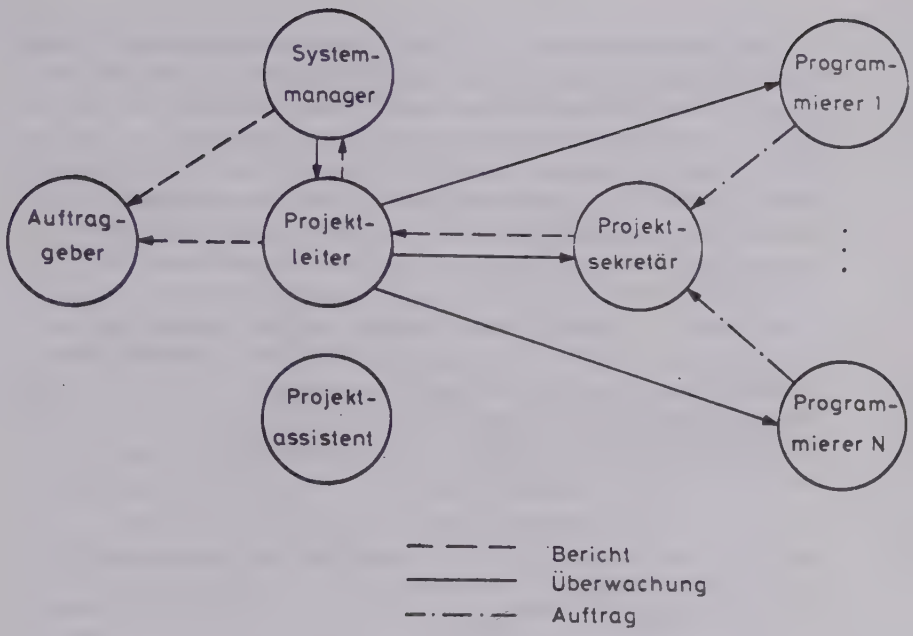


Bild 3.3.1 Struktur des Chief-Programmer-Team-Managements nach Baker und Mills

Urlaub) das technische Management übernehmen kann.

Der Projektsekretär (Programming Secretary) ist zuständig für die gesamte anfallende Verwaltungsarbeit im Projekt. Seine Hauptaufgabe ist somit die Führung einer Projektbibliothek, einer Datensammlung, die den aktuellen Projektstand wiedergibt. Die Projektbibliothek enthält zu jeder Zeit die letzten beiden Versionen aller enstandenen Quellprogramme, deren Maschinenprogramme und Dokumentation und weitere Projektdaten wie Testberichte u.ä.
Alle Rechenabläufe erfolgen im Chief-Programmer-Team-Konzept über den Projektsekretär, die Programmierer selbst haben also keinen Zugang zum Rechner. Statt dessen fordert jeder Programmierer jeden seiner Jobs beim Projektsekretär an und erhält auch seine Ausgaben über ihn zugestellt. Damit ist der Projektsekretär in der Lage, aus den ihn passierenden Ausgabelisten den aktuellen Arbeitsstand jedes Programmierers zu entnehmen. Diese von sich aus schon zuverlässigeren Daten als mögliche Berichte von

den Programmierern selbst werden von ihm gesammelt, aufbereitet und dem Projektleiter mitgeteilt. Über diesen Weg ist also auch der Projektleiter immer über dem aktuellen Stand des Projekt genau informiert und kann wiederum den Systemmanager qualifiziert unterrichten.
Zentrale Bedeutung kommt bei dieser Struktur der Prozedur zu, die jedes Modul durchläuft. Nach seiner Erstellung und dem Test durch den beauftragten Programmierer wird jedes Modul vom Projektsekretär formal freigegeben. Damit steht die Komponente in der Regel etwa zwei Wochen dem gesamten Team zur Verfügung und sollte von jedem Programmierer bei weiteren eigenen Testläufen mitverwendet werden. In einer Abnahmediskussion (walk through), an der alle technischen Teammitglieder teilnehmen, werden offengebliebene Fragen zur Funktion der neuen Komponente geklärt, es werden auftretende Fehler gesammelt und, wenn nötig, Kritik geübt.
Die Diskussion endet mit der formalen Abnahme des neuen Programmteils oder mit deren Rückgabe an den Programmierer zur Korrektur.

Der große Vorteil der Managementstruktur nach Baker und Mills liegt somit durch dauernde Kontrolle in der Vermeidung einer Situation, in der das Projekt automatisch abzulaufen scheint und bei der die Fehler erst zum Projektende sichtbar werden.

Wohlauf, lasset uns herniederfahren und ihre Sprache daselbst verwirren, daß keiner des anderen Sprache verstehe.
Also mußten sie aufhören, die Stadt zu bauen. Daher heißt ihr Name Babel, daß der Herr daselbst verwirrt hatte alle Länder Sprache und sie zerstreut von dort in alle Länder.

Die Bibel, 1. Mose 11

4. Engpässe der Softwareproduktion

Während sich der Text bisher darauf beschränkte, eine Beschreibung des aktuellen Zustands zu liefern und eine Standortdefinition durchzuführen, sollen im folgenden die nach Meinung des Verfassers kritischsten Stellen der Softwareproduktion besprochen und analysiert werden.

4.1 Produktion und Kommunikation

Produktion von Software ist die durchzuführende Aufgabe, aber Produktion in Teamarbeit führt ohne eine gut funktionierende Kommunikation zum Mißerfolg. So scheint schon dem Alten Testament ein aus diesem Grund gescheitertes Projekt entnehmbar: der Turmbau zu Babel im Zweistromland zwischen Euphrat und Tigris um 2000 vor Christus. Besteht nicht eine frappierende Ähnlichkeit zwischen der Oberlieferung des Chronisten und Situationen in heutigen Softwareprojekten, in denen bei den Teammitgliedern gegen Projektende die Vermutung aufkommt, daß "keiner des anderen Sprache verstehe"?.

Wird ein Gesamtprojekt zum Zwecke der schnelleren Fertigstellung in Teilaufgaben aufgespalten, so ergeben sich zwangsläufig Schnittstellen zwischen den Einzelkomponenten. Durch geschickte Aufteilung des Projekts (z.B. hierarchische Struktur) läßt sich zwar Einfluß auf die Anzahl und die Komplexität dieser Schnittstellen nehmen, eine prinzipielle Vermeidung von Schnittstellen läßt sich jedoch nicht erreichen. So besteht die Notwendigkeit, sich vor Produktionsbeginn über den Aufbau der Schnittstellen zu einigen, damit nicht nur die Teilaufgaben für sich zufriedenstellend arbeiten, sondern auch in ihrer Zusammenarbeit das geplante Gesamtergebnis liefern. Die Überlieferung des Turmbaus in Babylon läßt nach diesen Gedanken den Schluß zu, daß dieses Projekt der Antike nicht etwa aus Mangel an Arbeitskräften, sondern an ungenügender Absprache der Bauteilbereiche scheiterte.

Kommunikation ist zwangsläufig unproduktiv, sie trägt nicht direkt zum Projektfortschritt bei. Betrachtet werde der Einfluß der Kommunikation auf die Projekterstellungszeit T.

Im Idealfall tritt keine Kommunikation auf, T besteht nur aus dem Anteil Produktionszeit T_{prod}. Damit ist T in weiten Grenzen umgekehrt proportional zur Zahl der Mitarbeiter. Die Projekterstellungszeit ist überschaubar und jederzeit durch die Anzahl der Mitarbeiter beeinflußbar.

Die unvermeidliche Kommunikationszeit scheint zunächst aus einer rein additiven Konstanten T_{Komm} zu bestehen. Dieser erste Gedanke erweist sich jedoch bei näherer Überlegung als Trugschluß. Durch die wachsende Anzahl der Kommunikationskombinationen ist T_{Komm} stark von der Mitarbeiterzahl abhängig. Bespricht sich im Extremfall jeder Mitarbeiter mit jedem, so ergibt sich nach der Kombinationentheorie ein additiver Wert von $\binom{n}{2} \approx \frac{n^2}{2}$.

Bild 4.1.1 zeigt qualitativ die Abhängigkeit von Systemerstellungszeit und Zahl der Mitarbeiter ohne Kommunikation und in der Realität.

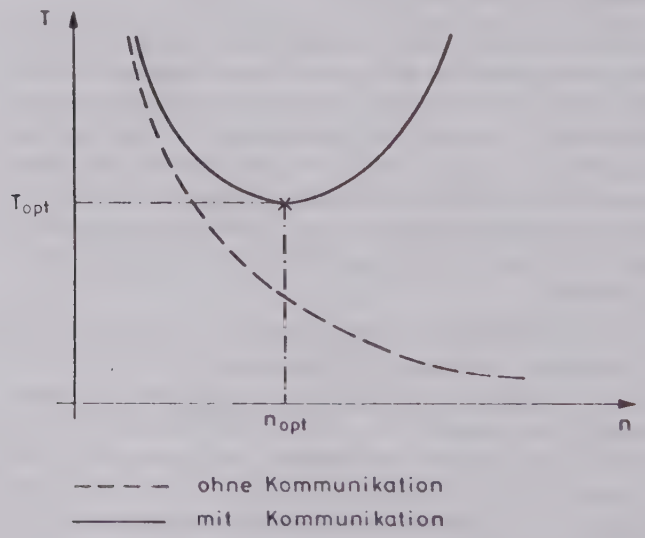


Bild 4.1.1 Zusammenhang zwischen der Systemerstellungszeit T und der Anzahl der Mitarbeiter n ohne und mit Kommunikation

Stark abhängig von der tatsächlichen Kommunikationszeit in einem konkreten Projekt gibt es ein Optimum für Zeit und Bearbeiteranzahl, das nicht durch die Erhöhung, speziell nicht durch die nachträgliche Erhöhung der Mitarbeiterzahl zu verbessern ist. Im Gegenteil steigen in diesem Fall, bedingt durch Einarbeitung, die Kommunikationszeiten rapide und Folge ist eine auch in der Praxis zu beobachtende Verschlechterung der Zeiteinteilung.

Fazit dieser Überlegungen ist damit die Erkenntnis, daß der Zusammenhang zwischen Systemerstellungszeit und Bearbeiteranzahl komplexer ist als erwartet. Dies führt konsequenterweise zu einer Einschränkung des Begriffs "Mannjahr".

Ein oft benutzter Erfahrungswert schreibt einem Programmierer eine - über den gesamten Softwareproduktionszyklus gerechnete - Produktivität von zwei bis drei Anweisungen pro Stunde zu. Daraus wird der Mannmonat (MM) zu 21 Arbeitstagen mit etwa 400 Anweisungen und das Mannjahr (MJ) zu 11 Arbeitsmonaten mit grob 4000 Anweisungen festgelegt.

Eine unzulässige Anwendung dieser Definition ist jedoch nach Ermittlung der Mitarbeiterzahl aus dem geschätzten Programmumfang die beliebige Umrechnung von Mitarbeiterzahlen in Produktionszeiten. Diese Umrechnungen setzen wieder das rein umgekehrt proportionale Verhältnis ohne Kommunikation voraus, das Bild 4.1.1 zeigt. Die Grundlage der Schätzung erfolgte jedoch im Optimum des realen, eben nicht umgekehrt proportionalen Mitarbeiter-Erstellungszeit-Verhältnisses, in dem eine Umrechnung bei Linearisierung nur in engen Bereichen statthaft ist.

Damit ergeben sich folgende Aussagen zur Optimierung des Kommunikationsproblems:

- die unbedingt nötigen Schnittstellen zwischen getrennt zu erstellenden Systemteilen sind durch geeignete Planung zu minimieren
- die geplanten Schnittstellen sind frühzeitig bis ins einzelne zu entwerfen und nur unter zwingenden Umständen zu modifizieren
- die entworfenen Schnittstellen sind jedem Programmierer frei zugänglich, aber nicht frei änderbar, aufzubewahren.

4.2. Das Dokumentationsproblem

Unter der Gesamtüberschrift "Kritik" ist es nötig, auch zum Thema Dokumentation einige Anmerkungen zu machen.

In Projekten der etablierten Ingenieurdisziplinen ist Dokumentation in Form von Bauzeichnungen, Schaltbildern, Meßpunkten usw. selbstverständlich. Nicht so in der Softwareproduktion. Bei den meisten Softwareprojekten ist die Dokumentation unvollständig, oft fehlt sie sogar ganz. Folgen fehlender Dokumentation zeigen sich oft erst zu Zeiten, in denen nicht mehr auf das Erstellerteam zurückgegriffen werden kann und sich niemand mehr an Aufbau und Arbeitsweise erinnert. So trifft Denert (/DEN 80/) die Feststellung: "Nicht dokumentierte Software ist unbrauchbar und wertlos", und tatsächlich ist schon manches Softwareprojekt vollkommen neu in Angriff genommen worden, nachdem die Feststellung getroffen wurde, daß die Einarbeitung eines neuen Teams in das bestehende System und dessen anschließende Korrektur oder Erweiterung den gleichen Zeitaufwand erfordern würde wie dessen Neuerstellung.

Im folgenden seien zunächst einige Gründe analysiert, die zum Problem mangelnder Dokumentation beitragen.

Es gibt keine einheitlichen Vorstellungen in der Fachbranche Softwareproduktion, von welcher Art und von welchem Umfang die zu erstellende Dokumentation sein sollte. Nur in Teilbereichen gibt es oft angewandte Standards wie z.B. die schon beim Thema Realisierung beschriebenen und in DIN genormten Programmablaufpläne. Viele Firmen haben ihre eigenen voneinander abweichenden Dokumentationsrichtlinien, aber auch diese decken zumeist nur Teilbereiche, z.B. die Verwendung von Kommentaren in Quellprogrammen ab.
Dokumentation wird heute im allgemeinen manuell, also ohne unterstützende automatisierende Hilfsmittel, erstellt. Der Vorgang der Dokumentation besteht also im wesentlichen aus der freien Formulierung verbaler Texte, deren schriftlicher Niederlegung und deren mehr oder weniger sortierten Sammlung in einem Ordnersystem durch den Programmierer selbst oder durch eine beauftragte Person. Verschärfend kommt hinzu, daß diese freie Formulierung vielen Programmierern schwerfällt, sie diese Tätigkeit nicht als ihre eigentliche Aufgabe ansehen und ihnen somit jede Motivation fehlt, diese ungeliebte Aufgabe zur Zufriedenheit durchzuführen.

Dokumentation wird in vielen Softwareprojekten in Form der nachträglichen Dokumentation durchgeführt. Bei der zeitlichen Abschätzung des Projekts und der Einteilung der Mitarbeiter wird nach geplanter Projektfertigstellung ein Zeitraum festgelegt, in dem ausschließlich die Dokumentation zu erstellen ist. Jedoch wird durch nicht kalkulierte Verzögerungen im Projekt im Regelfall während der vorgesehenen Dokumentationszeit noch produziert, der Projektfertigstellungstermin rückt näher und ein weiteres Projekt ist vielleicht in Vorbereitung. So dient manchem Projektleiter die Dokumentationszeit als willkommener Puffer für die Fertigstellung des Systems selbst. Die Dokumentation wird schon noch irgendwann erfolgen, wenn mal wieder Zeit für unproduktive Tätigkeiten ist. Daß die durch diese Maßnahmen in Umfang und Qualität stark eingeschränkte Dokumentation verhängnisvolle Folgen hat, wird in diesem Moment außer acht gelassen.

Über den generellen Inhalt von Softwaredokumentation haben sich eine Reihe Autoren Gedanken gemacht (z.B. /WAL 72/, /GIL 78/, /HEI 79/, /ALB 80/). Bild 4.2.1 zeigt eine Aufteilung der Dokumentationsarten nach ihrem speziellen Inhalt. Die Spezifikationsdokumentation enthält das gesamte während der Spezifikationsphase entstandene Pflichtenheft, das ja im allgemeinen auch die Vertragsgrundlage bildet. Sie enthält also sowohl Managementinformation wie auch Fachinformation. Eigentlicher Bestandteil dieser Spezifikationsdokumentation ist die Produktdokumentation, die Nachschlagewerke für den Anwender. Ihrer Bedeutung wegen sind sie hier speziell hervorgehoben. Es empfiehlt sich, diese oft als Handbücher bezeichneten Dokumente während der Systemspezifikation vollständig und in aller Ausführlichkeit zu erstellen, als ob das System schon existiere. Diese Erstellung zwingt dazu, sich mit allen Systemeinzelheiten vertraut zu machen; weiterhin können sie zur Entscheidungsfindung beitragen, wenn in späteren Produktionsphasen Zweifel über die Arbeitsweise des Systems auftreten sollten.

Die Projektdokumentation enthält alle relevanten Fakten über den inneren Aufbau des produzierten Systems, insbesondere also über den Programmablauf und die Struktur der bearbeiteten Daten. Bei Erstellung des Projekts nach dem beschriebenen Phasenmodell entstehen sowohl während der Planungsphase wie auch während der Realisierungsphase Unterlagen, die direkt diese Dokumentation darstellen. So stellen z.B. der in der Planungsphase festgelegte hierarchische Aufbau des Systems, die vereinbarten Modulschnittstellen oder die in der Realisierungsphase entstehenden Nassi/Shneider-

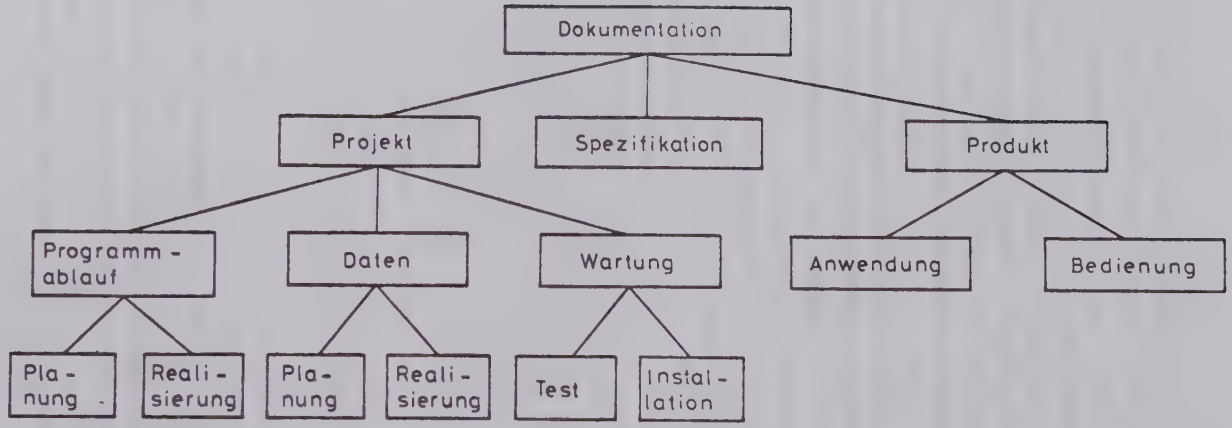


Bild 4.2.1 Dokumentationsarten nach ihrem Inhalt

man-Diagramme ausgezeichnete Unterlagen dar, die sowohl als Entwurfsgrundlage wie auch zur späteren Dokumentation dienen. Ein weiterer Vorteil dieser Vorgehensweise ist der Detaillierungsgrad der so entstehenden Dokumentation. Die Dokumentation eines Programmoduls detaillierter anzulegen als seine logische Struktur (z.B. in Nassi-Shneiderman-Technik) ist unsinnig, Sinn der Unterlagen ist ja gerade der Überblick. Detailliertere Informationen findet man dann bei Bedarf im Programmlisting selbst. Aus nachträglicher Dokumentation entstandene Unterlagen weisen oft gerade diesen Fehler auf, denn bei deren Erstellung ist der Programmierer gezwungen, seine fertigen Programme zu abstrahieren, eine Fähigkeit, die gerade dem Assemblerprogrammierer oft schwerfällt.

Einen letzten Teil der Dokumentationsunterlagen bilden die in der Testphase entstehenden Protokolle und Testrahmen. Sie dienen bei Installation und Wartung zur Kontrolle, wie ein aufgetretener Fehler, der ja offensichtlich seine Ursache in einer nie überprüften Datenkombination hat, entstehen konnte; erste Vermutungen lassen sich anhand dieser Unterlagen schnell auf ihre mögliche Stichhaltigkeit überprüfen.

Mit den so entstandenen und gesammelten Unterlagen hat sich das Problem der nachträglichen Dokumentation von selbst gelöst: es hat eine projektbegleitende Dokumentation stattgefunden. Allerdings sollten deutlich die Voraussetzungen dieser frappierend einfach klingenden Vorgehensweise genannt werden: jeder Programmierer muß sich genau an das Phasenmodell halten. Alle entstehenden Unterlagen müssen sauber und vollständig sein, um als Dokumentation verwandt zu werden. Es dürfen keinerlei "ad hoc"-Programme entstehen, die zuerst mal so hingeschrieben und vielleicht anschließend als Programmablaufplan gezeichnet werden. Nur so entsteht während des Projektablaufs eine verwendungsfähige, übersichtliche Dokumentation. Die Ergebnisse dieses Abschnitts ergeben damit folgende Thesen:

- Dokumentation sollte top-down, also mit zunehmender Verfeinerung, den Systemaufbau bis zur Ebene des logischen Modulablaufs darstellen.
- Dokumentation sollte projektbegleitend aus den Entwurfsunterlagen erstellt werden.
- Projektdokumentation sollte soweit möglich vereinheitlicht und vom Rechner unterstützt werden.

4.3 Die Teamorganisation

Es war zweifellos ein großer Schritt, als Baker und Mills mit dem Chief-Programmer-Management eine Organisationsform einführten, die die Erstellung von Software nach dem Phasenmodell optimal unterstützte. Die ersten Ideen zu diesem Konzept stammen jedoch vom Beginn der 70er Jahre, und sie beruhen unzweifelhaft auf der Voraussetzung folgender Fakten:

Die Softwareproduktion erfolgt mit einem batchorientierten größeren Rechner. Die Jobs der Programmierer erfolgen vorteilhaft über ein lochkartenstapelähnliches Medium; eine Idee der Organisationsform beruht auf deren Kontrolle durch Zentralisierung über den Projektsekretär.

Seit dieser Zeit sind zwei grundlegende Änderungen eingetreten. Zum einen hat ein fast vollständiger Übergang zu on-line-Systemen mit dem Vorteil des extrem verkürzten Editier-Compiler-Zyklus stattgefunden. Zum anderen hat sich gerade im Bereich der Mini- und Mikrorechner das Einplatz- oder kleine Mehrplatz-Entwicklungssystem durchgesetzt, an dem einem oder wenigen Programmierern während der Konsolsitzungszeit der gesamte Rechner zur Verfügung steht. In diesem Zusammenhang geht der Trend zu privaten Massenspeicher-Datenträgern wie Digital-Cassette und Floppy-Disk.

Dem Vorteil der mit diesen Entwicklungen verbundenen sehr viel effektiveren Arbeitsweise für den Programmierer steht der Nachteil der wieder eingetretenen Unkontrollierbarkeit entgegen. Da es selbstverständlich unsinnig ist, auf diesem Gebiet Rückschritte zu unternehmen, ist es an der Zeit, die Aufgabe und die Arbeitsweise des Projektsekretärs neu zu überdenken, um die Projektkontrolle nicht wieder zu verlieren.

Nach Abschnitt 3 ist die dem Projektsekretär zugedachte Arbeit in zwei Teilaufgaben aufzuspalten. Zunächst soll er die Programmierer von Routineaufgaben und Verwaltung entlasten. Gleichsam zunächst als Nebeneffekt ergibt sich dann durch seine zentrale Rolle die Möglichkeit, die ihn sowieso passierenden Unterlagen von ihm prüfen zu lassen und dem Projektleiter Bericht zu erstatten. Zur zweiten Aufgabe gehört allerdings wesentlich mehr Fachwissen als zu der reinen Verwaltung. Damit war die Rolle des Projektsekretärs sicher immer ein Engpaß. Seine zentrale Stellung zwischen Projektleiter und Programmierern verlangt nach einem qualifizierten Mitarbeiter, seine Qualifikation wird aber durch Aufbürdung vieler Routineaufgaben in Teilbereichen nicht genutzt.

Mit Obergang zu den heute üblichen On-line-Systemen entfallen automatisch viele Teile der Routinearbeit. Eine Aufgabe wie den expliziten Lauf eines Jobs gibt es dort einfach nicht mehr, und die mit einer vergleichenden Aufgabe, etwa einer Konsolsitzung, verbundenen Verwaltungstätigkeiten haben sich so dezimiert, daß sie getrost dem Programmierer selbst überlassen werden können.

Damit geht die Hauptaufgabe des Projektsekretärs automatisch mehr zu den qualifizierteren Tätigkeiten über, der Oberprüfung der Programmierer, der Sammlung ihrer Berichte und darin eingeschlossen, die Führung der Projektbibliothek. Seine Sekretärsaufgaben sind also eher Koordinierungsaufgaben gewichen.

Natürlich muß der Projektsekretär zur Erfüllung dieser Aufgaben wiederum einen genauen Oberblick über Aufgabe und Entwicklungsstand jedes Programmierers haben. Was früher automatisch durch die Verwaltung der Outputs geschah, kann heute durch eine regelmäßige Kopie des Arbeitsdatenträgers jedes Programmierers erfolgen. Die Analyse seines Inhalts bringt den Projektsekretär unbeeinflußt auf den gewünschten Kenntnisstand. Bestimmte Aufgaben verwaltungstechnischer Art, wie z.B. die eigene Einschätzung seines Entwicklungsstands durch den Programmierer, sind sicher formalisierbar. Ebenso regelmäßig, wie der Projektsekretär Kopien aller Arbeitsdatenträger erhält, sollten diese Kopien aktualisiert und an die einzelnen Bearbeiter zurückgegeben werden. Bei diesem Datentransfer werden beispielsweise neu freigegebene Objektmodule zur weiteren Verwendung übergeben oder Systemneuerungen auf mehr oder weniger formale Weise mitgeteilt. Nach dieser Idee stehen alle Programmierer mit dem Projektsekretär in einem regelmäßigen sternförmig organisierten Datenträgeraustausch. Die Auswertung und Aufbereitung dieser Datenträger muß durch den Projektsekretär am Rechner selbst geschehen und kann durch Einsatz einer unterstützenden Software halbautomatisiert werden. Diese Automatisierung beinhaltet einen weiteren Vorteil: Der Arbeit des Projektsekretärs wird etwas von ihrer Unentbehrlichkeit genommen. Der Engpaß bei Ausfall dieser Figur in der Organisation wird dadurch gemildert, daß die rechnerunterstützten Aufgaben auch kurzfristig ersatzweise z.B. durch den Projektassistenten durchgeführt werden können.

Als Ergebnis läßt sich also zusammenfassend festhalten:

- Die Arbeit des Projektsekretärs verlagert sich im Zuge der On-line-Systeme weg von Routineaufgabenbelastung hin zu Koordinierungs- und Kontrollaufgaben.
- Die damit verbundenen bibliothekarischen und verwaltungstechnischen Aufgaben können am Rechner selbst durchgeführt und halbautomatisiert werden.

"Dieser Weg?" sagte Frodo.
"Ja, Herr!" sagte Sam Gamdschie, "das scheint mir ein guter Weg zu sein, wenn Du verstehst, was ich meine."

J.R.R. Tolkien: Der Herr der Ringe

5. Eine Beschreibungsbank für die Softwareproduktion

Der folgende Abschnitt enthält die grundsätzlichen Überlegungen an ein umfassendes, den Projektsekretär unterstützendes Softwaresystem: der Softwarebeschreibungsbank. Ausgehend von diesen Überlegungen werden die Anforderungen an ein solches System formuliert.

5.1 Eine Standortanalyse

5.1.1 Voraussetzung an die Systemumgebung

Die Absicht, ein für welche Aufgaben auch immer bestimmtes Softwaresystem zu bauen, setzt, wie bereits beschrieben, die möglichst exakte Analyse des Systemeinsatzgebiets, also der späteren Systemumgebung voraus. Die Absicht, ein Softwaresystem zur Unterstützung der Softwareproduktion selbst und speziell des Projektsekretärs zu erstellen, zwingt also dazu, die Softwareproduktionsstätte zu definieren und bedingt auch die Festlegung der durch Hardwarebeschränkungen gegebenen Grenzen des Beschreibungsbankeinsatzgebiets.

Erste Voraussetzung an die Softwareproduktionsstätte ist ohne Zweifel deren personelle Organisation nach dem beschriebenen Baker und Mills'schen Chief-Programmer-Team. Die konkret zu unterstützende Figur ist der Projektsekretär, dessen Hauptaufgabe die Führung der Beschreibungsbank wird. Eine Anforderung an die Qualifikation des Projektsekretärs ist damit die Bedingung, daß er mit der Wirkungsweise und Bedienung des oder der zur Verfügung stehenden Rechner vertraut ist.

Um seine festgelegten Aufgaben mit Hilfe der Beschreibungsbank durchführen zu können, sind die ersten zwei Forderungen aus Sicht des Projektsekretärs zu stellen:

(1) Die Beschreibungsbank muß die Projektbibliothek integrieren, also Quellprogramme, Maschinenprogramme und Dokumentation beinhalten.

(2) Die Beschreibungsbank muß Managementinformation in Form von Kalenderdaten für geplante und eingetretene überprüfbare Ereignisse integrieren.

Eine weitere Voraussetzung werde für die Hardwareumgebung der Produktionsstätte getroffen. Einsatzgebiet der Beschreibungsbank seien Einplatzentwicklungssysteme oder kleine Mehrplatzentwicklungssysteme auf der Basis von Mikrorechnern. Durch die wachsende Bedeutung der 8 Bit- und 16 Bit-Mikrorechnersysteme und die ständig steigende Zahl der Anwendungsgebiete ist gerade hier die Softwareproduktion besonders groß. Systeme von 40K byte Umfang und mehr sind heute keine Seltenheit mehr. Auf der anderen Seite ist gerade hier die Programmierunterstützung und speziell die Verwaltungsunterstützung schlecht und verbesserungsbedürftig.

Mit dieser Voraussetzung ist gleichzeitig der Entwicklungsrechner spezifiziert. Typisch ist ein Ein-Platz-Rechnersystem mit Konsolsichtgerät, angeschlossenem Schnelldrucker und Billigmassenspeicher. Das System ist im allgemeinen mit flüchtigen Halbleiterspeichern (RAM's) ausgestattet. Jeder Programmierer betreibt das System mit seinem eigenen Massenspeicherdatenträger, vorzugsweise Floppy-Disk, der ein einfaches Betriebssystem und Systemoperatoren wie Editor, Assembler und Compiler enthält, und auf dem er auch seine eigenen Daten wie Quellprogramme und Objekte speichert.

Die Versorgung der Beschreibungsbank soll nach dem Vorschlag aus Abschnitt 4.3 erfolgen. Regelmäßig werde eine Kopie jedes Arbeitsdatenträgers dem Projektsekretär zugestellt, von ihm bearbeitet und aufbereitet und anschließend zurückgesandt. Es erscheint vorteilhaft, speziell für die zu erstellenden Dokumentationen, den einzelnen Programmierer diese Arbeiten geeignet vorbereiten zu lassen. Der Projektsekretär wird entlastet, indem er diese Unterlagen dann nur noch maschinell sammelt und aufbereitet; der Programmierer wird gezwungen, frühzeitig und - das werde später genau überdacht - formal die von ihm zu erstellenden Programmteile zu dokumentieren. Zwei weitere Forderungen an die Beschreibungsbank werden somit durch die geplante Einsatzform gestellt:

(3) Das Beschreibungsbanksystem muß auf den gleichen Rechnertypen lauffähig sein, auf denen das zu beschreibende Softwareprojekt entwickelt wird.

(4) Jeder Programmierer muß auf die Teile der Beschreibungsbank schreibend zugreifen können, die seine Arbeit betreffen. Der Projektsekretär hat schreibenden Zugriff auf alle Teile der Beschreibungsbank.

Das bedeutet, daß zumindest Teile der Beschreibungsbank als Duplikat auf einem privaten Datenträger jedes Programmierers untergebracht sein müssen und erfordert auf der anderen Seite einen Schutz des Systems selbst gegen unberechtigt durchgeführte Manipulationen seiner verwalteten Daten. Unter allen Umständen müssen trotz mehrfach vorhandener Daten Doppeldeutigkeiten vermieden werden.

Eine weitere Oberlegung zeigt, daß zumindest der lesende Zugriff auf die Beschreibungsbank durch jeden Programmierer so wenig wie möglich eingeschränkt werden sollte. Die zur Dokumentation verwalteten Daten enthalten Information über den Aufbau des gesamten in Arbeit befindlichen Systems, speziell über Modulaufgaben, Schnittstellen, gemeinsame Daten usw.. Bei Bedarf kann sich also jeder Programmierer über ihn betreffende und von Kollegen bearbeitete Teile des Systems informieren, ohne diese selbst anzusprechen. Ziel und Nutzen dieser Möglichkeit wäre die Verringerung der Kommunikation und damit die Erhöhung der Produktivität. Damit ist formulierbar:

(5) Jeder Programmierer muß lesend auf alle Teile der Beschreibungsbank zugreifen können.

Die Unterstützung durch die Beschreibungsbank beginnt in der Planungsphase und endet für die fachlichen Daten mit der Modulrealisierung; für die verwaltungstechnischen Daten können sich durchaus noch Modifikationen während der Systemwartung ergeben. Die Beseitigung von Fehlern durch die Wartung führt zu Korrekturen und damit zu neuen Versionen einzelner Module. Es ergibt sich dann die Forderung, einzelne Modulbeschreibungen zu aktualisieren und bei Bedarf die alte Version zu archivieren. Im Laufe der Softwarelebensdauer sollte also deren Beschreibungsbank mehrere Versionen verwalten können und gleichzeitig eine Obersicht über deren Entstehung führen. Es ergibt sich die Forderung:

(6) Die Beschreibungsbank unterstützt das beschreibende Softwaresystem von der Planungsphase bis zur Obergabe an den Kunden. Dann wird die Beschreibungsbank zum Beschreibungsarchiv, das zusätzlich die im Rahmen der Wartung entstehenden neuen Versionen verwaltet.

Ziel der Beschreibungsbank sei die Verwaltung von Projekt- und Systeminformation, nicht aber die Unterstützung von Spezifikations- oder Entwurfsmethoden. Dies hat vor allem zwei Gründe.

Zunächst gilt, daß die bekannten Verfahren (z.B. Spezifikationssprachen, HIPO-Technik, Jackson-Verfahren usw.) zum Teil noch in der Entwicklung, zum Teil umstritten und zum Teil nur für bestimmte Anwendungsgebiete einsetzbar sind. Jedenfalls erscheint es zum gegenwärtigen Zeitpunkt unsinnig, eine oder mehrere dieser Methoden vorzuziehen und vorzuschreiben. Die Durchführung von Spezifikation, Entwurf und Realisierung bleibe nach Auffassung des Verfassers also weiterhin der Intuition des Menschen vorbehalten. Ihm bleibt es natürlich unbenommen, eine derartige Methode einzusetzen, wenn er sie für sinnvoll erachtet. Eine aus der Praxis zu beobachtende Tatsache ist jedoch mangelnde Akzeptanz beim potentiellen Anwender wegen Unanschaulichkeit und starken Formalismus, ein Problem, das sich sicher bei Implementierung auf einem Rechner durch weitere unvermeidbare Einschränkungen verschärfen würde. Als Beispiel seien alle zeichnerischen Methoden genannt; ihre im allgemeinen relativ gute Anschaulichkeit wird bei Rechnerimplementierung ohne die Verwendung eines graphischen Sichtgeräts stark eingeschränkt.

Der zweite Grund ist profanerer Art. Mit dem vorgesehenen Einsatzgebiet der Beschreibungsbank auf kleinen Rechnersystemen gibt es von der physikalischen Seite her (Wortbreite, Speicherplatz, Rechenzeit usw.) zu berücksichtigende Einschränkungen für die Implementierung der genannten Methoden. Im Rahmen von Oberlegungen zum Implementierungssystem wird darauf näher eingegangen werden.

Der Umfang von Software-Entwicklungssystemen wie PASILA (/BLA 81/), EPOS (/BIE 79/) oder anderer schon genannter Systeme wird also bewußt nicht angestrebt, der Schwerpunkt liegt vielmehr auf der Verwaltung der Projektdaten. Diese Voraussetzung bedeutet auf der anderen Seite keine Einschränkung für eine mögliche Verifizierung der zu verwaltenden Daten. Es wird näher zu überprüfen sein, welche Informationen automatisch verifizierbar sind, speziell unter dem Gesichtspunkt, daß voneinander unabhängig einzelne Programmierer Eintragungen in verschiedene Exemplare der Beschreibungsbank machen dürfen.

Eine letzte Forderung stellt damit eine Einschränkung dar:

(7) Hauptaufgabe der Beschreibungsbank ist die Verwaltung der Projektdaten, nicht die Unterstützung von Methoden. Nicht eingeschränkt ist damit die weitestgehende Verifikation der abgespeicherten Daten.

5.1.2 Die Akzeptanz durch die Benutzer

Der Einsatz neuer Techniken, Verfahren und Methoden bewirkt häufig eine Reaktion, die zu wenig bedacht wurde: die vollständige oder teilweise Ablehnung durch die Benutzer. Für sie existiert zunächst nur eine oft kompliziert erscheinende Neuerung, die sie aus ihrer für gut gehaltenen Gewohnheit reißt und die von ihnen die Einarbeitung in eine neue Bedienung verlangt. Im Gegensatz dazu sind sie oft nicht in der Lage oder wollen wissentlich nicht die objektiven Vorteile erkennen, die den Entschluß zur Einführung der Neuerung bewirkt haben. So ergibt sich eine pessimistische Grundstimmung, die bewirken kann, daß bei einem nichtigen Anlaß, z.B. ein Fehler in der Neuerung psychologisch erklärbare Abwehrhaltungen auftreten.

Das neue System, in diesem Fall die Beschreibungsbank, sollte also in einer Art und Weise eingeführt werden, die diese pessimistische Reaktion zu vermeiden sucht und eine positive Grundstimmung fördert. Oberlegungen dieser Art basieren sicher stark auf psychologischen Grundsätzen und sollen hier nur am Rande erwähnt werden; es ist jedoch schon Sache der Beschreibungsbankentwickler, für die Akzeptanz durch die späteren Benutzer zu sorgen, also die pessimistische Stimmung nicht wieder aufkommen zu lassen.

Damit ist es sinnvoll, vor der Erstellung des Systems gezielte Oberlegungen zu seiner Akzeptanz anzustellen. Dabei geht es natürlich weniger um die selbstverständliche Einführung eines möglichst fehlerfreien Systems, als vielmehr um

- die Arbeitsweise des Benutzers mit dem System
- die Auswirkungen des Systems auf die Arbeit des Benutzers.

Ein wesentlicher Gesichtspunkt bei der Einführung einer koordinierenden Beschreibungsbank scheint die drohende Entfremdung des einzelnen Programmierers vom Gesamtprojekt zu sein. Ihm darf nicht das Gefühl vermittelt werden, daß man seine Initiative einschränkt und er nur noch einen kleinen Teil eines unüberschaubaren Projekts bearbeitet. Damit ist der Gedanke, jedem Programmierer den lesenden Zugriff zu der vollständigen Systembeschreibung zu ermöglichen, auch aus dieser Blickrichtung sinnvoll. Aber die Erlaubnis dieses - ja aktiv durchzuführenden - Zugriffs sollte nicht als ausreichend angesehen werden. Die Beschreibungsbank ersetzt nicht die regelmäßig durchzuführenden Team-Versammlungen. Die Kommunikationszeit unter allen Umstän-

den minimieren zu wollen, führt sicher zu unübersehbaren anderen, die Produktivität negativ beeinflussenden Faktoren.

Das Isolationsproblem ist ein zweiter Faktor in diesem Zusammenhang. Ein Programmierer, der weiß, daß ihm zur Information über eine Schnittstelle als einzige Möglichkeit die Beschreibungsbank zur Verfügung steht, kann sich abgekapselt fühlen. Beim Konzept der Beschreibungsbank ist also zu berücksichtigen, daß trotz dafürsprechender Gründe die Kommunikation als Grundlage der Teamarbeit nicht ganz wegfällt. Insbesondere sind sicher Verfahren nach Art der "Mail-box" zu vermeiden, bei denen ein Programmierer etwa eine Nachricht an seinen Kollegen in schriftlicher Form über eine Beschreibungsbankverwaltung weitergeben muß. Vielmehr ist es sinnvoll, die Kommunikation in gezielte Bahnen zu lenken, ihre Ergebnisse in der Beschreibungsbank abzulegen und zu verifizieren.

Ein zweiter Punkt ist der direkte Kontakt des Benutzers mit dem System, im allgemeinen unter dem Stichwort "Benutzerfreundlichkeit" zusammengefaßt.

Ein beschreibungsbankähnliches System muß eine grundsätzlich dialogorientierte Arbeitsweise unterstützen. Die Arbeit mit dem System besteht dann aus einem "Gespräch", bei dem je nach Aufgabe der Mensch oder der Rechner die aktive, führende Rolle übernimmt. Drei Punkte, die die Qualität eines Dialogs verbessern, sollen speziell herausgehoben werden.

- Die Arbeit mit dem System muß übersichtlich und anschaulich sein.
  Dieses Kriterium kann auf verschiedene Teilgebiete der Beschreibungsbank angewandt werden. Zunächst ist eine logisch sinnvolle Struktur der zu verwaltenden Daten aus Sicht des Benutzers Voraussetzung. Weiterhin müssen die Operationen, die auf diese Daten anwendbar sind, klar und einfach sein. Schließlich ist das äußere Bild des Dialogablaufs von entscheidender Bedeutung. Besonders bewährt hat sich in diesem Zusammenhang bei geeignetem Konsolmedium (Sichtgerät mit steuerbarer Schreibmarke) eine Menü-Technik. Sie ermöglicht es dem Benutzer durch schrittweises Vorgehen, Auswahl nach vorgelegten Entscheidungsmöglichkeiten und Beschreiben vom Rechner vorbereiteter Sichtfensterbereiche Daten einzutragen,

zu korrigieren, abzufragen und bearbeiten zu lassen. Dies beinhaltet speziell eine unter allen Umständen klare Rechnerreaktion; Ausgaben wie "Error 31" tragen nicht zur Akzeptanz bei.

- Die Benutzereingaben sollten so früh wie möglich auf Konsistenz geprüft werden.
  Der unabsichtlich aufgetretene Eingabefehler ist eine typische Folge der menschlichen Unvollkommenheit. Werden diese Fehler nicht oder erst nach einer unakzeptabel langen Zeit maschinell erkannt und führen dann vielleicht zur Rückweisung der gesamten Eingabe, so resultieren daraus Unlust und Frustration. Eine Überlegung zeigt, daß es für den Fall der Beschreibungsbank vier Fehlertypen gibt, die zu unterschiedlichen Zeitpunkten frühestmöglich erkannt werden können.

  Es sind die folgenden:

  Der einfache lexikalische Fehler ist mit dem Eingabezeichen selbst erkennbar; typisches Beispiel ist ein Buchstabe in einer Dezimalzahl. Bei seinem Erkennen wird vorteilhaft das fehlerhafte Zeichen weder akzeptiert noch protokolliert. Statt dessen erfolge z.B. eine akustische Warnung.

  Der sich aus dem Zusammenhang einer Eingabeeinheit ergebende Fehler ist nach Abschluß deren Eingabe erkennbar. Eine solche Einheit sollte auch im Extremfall nicht länger als 50 Zeichen sein. Der Fehler werde durch einen Text gemeldet und der Abschnitt erneut zur Eingabe angefragt; eine Fortsetzung der Bearbeitung werde erst nach richtiger Eingabe zugelassen.

  Die Behandlung von Fehlern, die sich aus dem Zusammenhang eines Eingabeabschnitts ergeben, ist umstrittener. Es ist unsinnig, die gesamte Eingabe, die durchaus aus mehreren hundert Eingabeeinheiten bestehen kann, zu verwerfen. Also müssen Teile der Information korrigierbar sein und die Fehlerüberprüfung muß wiederholt werden. Da jedoch Fehler im Kontext oft schwer erkennbar sind und ein Beharren auf deren Korrektur den flüssigen Ablauf beeinträchtigen könnte, ist es sinnvoll, auch die Ablage fehlerhafter Daten mit einem entsprechenden Vermerk zuzulassen. Die Korrektur erfolgt dann später, wenn der Grund des Fehlers erkannt wurde.

Eine spezielle Fehlerart tritt bei der geplanten Arbeitsweise mit der Beschreibungsbank auf. Es sind Fehler denkbar, die erst beim maschinellen Vergleich der Eintragungen zweier unabhängiger Programmierer aufgedeckt werden. Hier erscheint es unsinnig, dem einen oder dem anderen Datensatz den Vorzug zu geben, vielmehr sollte für alle Beteiligten der alte Zustand, vielleicht auch der Zustand der leeren Eingabe, wiederhergestellt werden. Ein Protokoll sollte auf die Unstimmigkeiten hinweisen und so deren Klärung fordern.

- Die Rechnerreaktion sollte der Dialogarbeitsweise angepasst sein.
  Die Praxis zeigt, daß sich die Schnelligkeit der Rechnerreaktionen direkt auf die Zufriedenheit der Bearbeiter auswirkt. Das untätige Warten auf Systemantwort führt zu Nervosität und Unruhe. Da die Rechnerreaktion jedoch wegen unvermeidlicher Bearbeitungszeiten und Massenspeicherzugriffe in der Praxis oft nicht unmerklich klein gehalten werden kann, sei hier überlegt, wann welche Zeiten akzeptabel sind.

  Die Eingabe eines einzelnen Zeichens, dessen Oberprüfung und Protokoll sollte unter allen Umständen unmerklich schnell erfolgen. Nur so wird der Eingabefluß nicht gebremst. Die Reaktion auf eine Eingabeeinheit darf durchaus merklich sein; auch der Bearbeiter macht in diesem Moment eine Denkpause. Eine längere Pause als 1...2 Sekunden unterbricht jedoch seinen Gedankenfluß. Längere Reaktionen werden bei Abschluß eines Eingabeabschnitts vom Programmierer akzeptiert, hat er ja hier eine eigene aktive Phase abgeschlossen und dem Rechner die Initiative übertragen. Reaktionszeiten, die über 30 Sekunden hinausgehen, sollten jedoch wirklich umfangreichen Aufgaben wie komplizierten Suchläufen vorbehalten bleiben.

Werden diese Punkte im weiteren Verlauf der Oberlegungen beachtet, ist sicher eine gute Grundlage für die psychologische Seite des Beschreibungsbankeinsatzes gegeben.

5.1.3 Minimale Implementierungsrechnerkonfiguration

Die Analyse eines Beschreibungsbanksystems beinhaltet die Betrachtung und Festlegung von Daten, die das Implementierungsrechnersystem charakterisieren. Ziel der folgenden Oberlegungen ist die Definition von Anforderungen an dessen Architektur und Ausbau.

Ein erster Ansatz ist die Hardware des Einsatzsystems. Um bei einem residenten gehaltenen Betriebssystem und dem Code des Beschreibungsbanksystems selbst noch effektiv Datensätze zu bearbeiten, ist eine Untergrenze für den Zentralspeicherausbau von 64 K byte wünschenswert. Damit stößt man bei den meisten 8 Bit-Prozessoren auf der anderen Seite schon auf die Grenze der Adressierungsmöglichkeiten; ohne großen Aufwand ist also in diesem Fall keine Erweiterung möglich. Als Massenspeicherausbau des Systems sei mindestens ein Floppy-Disk-Laufwerk mit Double-Density-Aufzeichnungsverfahren vorausgesetzt. Die Speicherkapazität einer Diskette beträgt in diesem Fall etwa 0,7 M byte, die Zugriffszeit auf einen beliebigen Datensektor liegt bei maximal etwa 0.7 Sekunden. Wird ein Verwaltungsaufwand von 20% der Nutzdatenmenge veranschlagt, so ergibt eine erste grobe Schätzung bei 100 zu beschreibenden Modulen eine Reservierung von 4 K byte pro Beschreibung. Bei dieser selbstauferlegten Speichereinschränkung ist also vorauszusehen, daß die Ablage der Daten so effektiv und redundanzfrei wie möglich zu erfolgen hat.

Um das gesamte System nebst allen Quellen und Objekten zu verwalten, erweist es sich als Vorteil, das Rechnersystem, an dem der Projektsekretär seine Arbeiten erledigt, mit einem Fest- oder Wechselplattenlaufwerk auszustatten. Diese jedoch nicht als Bedingung, sondern als Empfehlung formulierte Erweiterung hilft, den Systemdurchsatz zu erhöhen und vermeidet den häufigen Wechsel von Datenträgern. Plattenlaufwerke sind heute, vorteilhaft auch in Winchester-Technologie, mit einer Kapazität von 7...20 M byte für fast alle Mikrorechner - Entwicklungssysteme zu erwerben; ihr aktueller Preis macht jedoch einen Einsatz vom Einzelfall abhängig.

Der Ausbau an peripheren Geräten soll ein Sichtgerät als Benutzerkonsole und einen schnellen Drucker umfassen. Das Sichtgerät erlaube zur Unterstützung der Menü-Technik die benutzergesteuerte Positionierung einer Schreibmarke. Der schnelle Drucker dient zur unbedingt notwendigen Erzeugung der Information in schriftlicher Form; um akzeptable Geschwindigkeiten zu erreichen, sollte die Druckgeschwindigkeit nicht unter 150 Zeichen/sec

liegen. Es ist in der Regel davon auszugehen, daß sowieso jedes effektive Entwicklungssystem mit diesen Geräten ausgestattet ist.

Bild 5.1.1 zeigt schematisch den festgelegten Aufbau um die Beschreibungsbank.
Der Projektsekretär-Rechner und die Programmierer-Rechner können physikalisch identisch sein; an ihnen wird auch die eigentliche Programmentwicklung durchgeführt.

Auch die Bedingungen an die Betriebssoftware sollen kurz betrachtet werden. Grundvoraussetzung an das zugrundeliegende Betriebssystem ist neben der Möglichkeit, Programme von Diskette zu laden und zu starten, die Verwaltung der auf der Diskette zu speichernden Daten in Dateien mit logischen Namen. Dies beinhaltet die Führung eines Inhaltsverzeichnisses und die Verwaltung des Disketten-Freispeichers. Dieses Dateimanagement muß über Systemaufrufe zur Verfügung stehen. Vorteilhaft wäre die Verwaltung der peripheren Geräte über die Dateiverwaltung, damit wäre z.B. der Schnelldrucker wie eine Datei mit einem speziellen Namen behandelbar.

Die Benutzbarkeit einer auf dem System lauffähigen Datenbank erscheint zunächst als großer Vorteil, denn es werden sicher recht viele Daten verwaltet werden müssen. Datenbanken stehen jedoch in der Regel für Mikrorechner-Entwicklungssysteme der geplanten Größe nicht zur Verfügung. Weiterhin ist zu befürchten, daß sich ihr Einsatz in Bezug auf Rechenzeit und Massenspeicherbedarf als zu unrentabel erwiese. Die Beschreibungsbankimplementierung wird Verfahren zur effektiven Ablage der von ihr verwalteten Daten enthalten müssen.

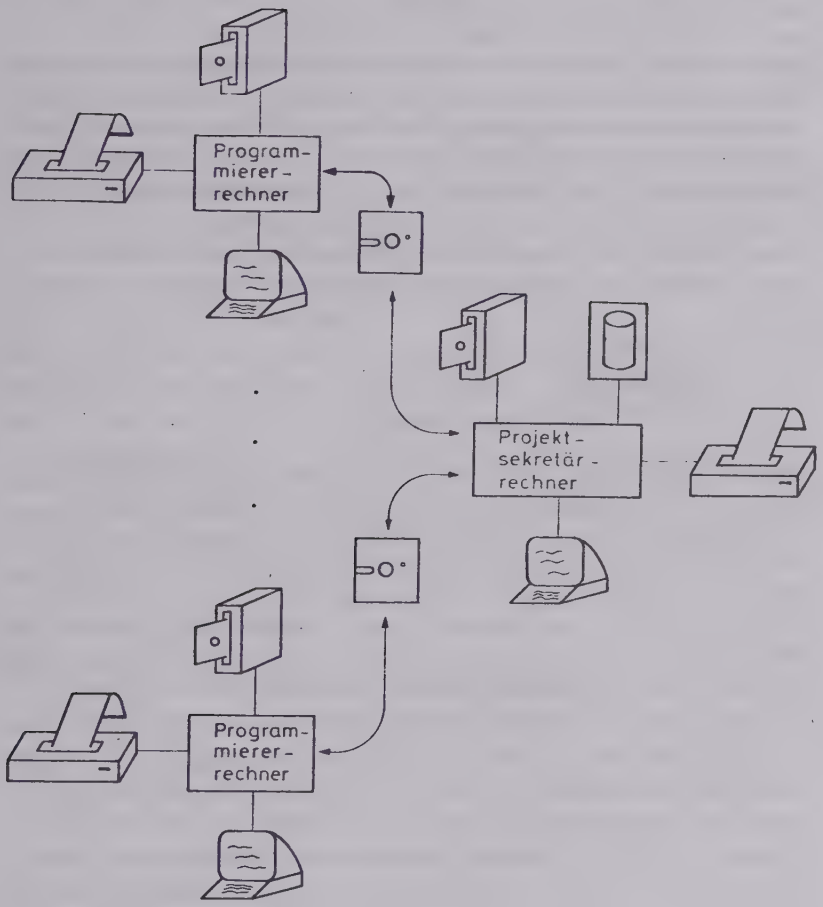


Bild 5.1.1 Umgebung und Arbeitsweise der Beschreibungsbank

5.2 Die Daten in der Beschreibungsbank

5.2.1 Klassifizierung der Informationsfülle

Beherrschendes Element einer Beschreibungsbank sind die von ihr verwalteten Daten. Ein Ansatz zur weiteren Spezifizierung sei also - im Gegensatz zum Vorgehen bei den meisten prozeßrechnerorientierten Programmsystemen - zunächst die Klassifizierung dieser Daten nach verschiedenen Kriterien und, daraus ableitbar, ihre Struktur und ihr Inhalt. Von diesen Ergebnissen ausgehend sollen dann die Operationen festgelegt werden, die auf diese Daten anwendbar sind. Es sollen drei Grundklassen von Daten festgelegt werden, die in der Beschreibungsbank enthalten sind:

- Managementdaten
- Verwaltungsdaten
- fachbezogene Daten.

Managementdaten machen Angaben über Einteilung und Verlauf des Projekts. Durch sie soll der Projektsekretär in der Lage sein, dem Projektleiter jederzeit über den genauen Stand des Projekts Bericht zu erstatten. Insbesondere von Nutzen sind Angaben über den Produktionsstand der einzelnen Module (in der Realisierung, getestet, freigegeben...) und über den Namen ihres Programmierers, und - daraus ableitbar - Angaben über Abweichungen von der geplanten Produktion. Nach Abschluß des Projekts sind die Managementdaten allenfallls zu Analysezwecken verwendbar, etwa einer Studie des Projektverlaufs mit dem Ziel, erkannte Organisationsfehler in Zukunft zu vermeiden.

Verwaltungsdaten stellen ein Inhaltsverzeichnis für die Quellen und Objekte des entstehenden Systems dar. Sie geben also etwa die aktuelle Versionsnummer der einzelnen Module und den Namen des Datenträgers an, auf dem sie abgespeichert sind. Auch nach Abschluß des Projekts sollten sie bei jeder Wartung aktuallisiert werden; sie führen dann Buch über aufgetretene Fehler, Ersetzungen von Modulen und deren Durchführung und archivieren alte Versionen.

Die fachbezogenen Daten schließlich beinhalten Information über den inneren Aufbau des Produkts. Sie stellen die physikalisch bei weitem umfangreichste Informationsmenge dar und überdecken den Bereich von der Struktur des Systemaufbaus bis hin zur logischen Ablaufbeschreibung der einzelnen Module. Sie dienen während der Produktion als verbindliche Grundlage über alle mehr als einen Programmierer betreffenden Vereinbarungen und stellen nach Produktübergabe die wichtigste Unterlage bei der Systemwartung dar. Aus diesem Einsatz resultiert die Forderung, daß die oft vorkommende Suche nach einer speziellen Stelle im System zunächst durch einen Überblick und erst anschließend durch eine Verfeinerung zu unterstützen ist. Ein Überblick ergibt sich vorteilhaft durch eine Beschreibung der Systemhierarchie, ein Einzelmodul wird zunächst durch seine Funktion und seine Schnittstellen und schließlich durch seinen inneren Aufbau beschrieben.

Die Klassifizierung und teilweise nähere Benennung der Daten läßt erkennen, daß zumindest große Teile formalisierbar und durch eine begrenzte Menge Begriffe ausdrückbar sind. Diesen Effekt ausnutzend sollte ein Ziel der folgenden Überlegungen sein, eine Art teilformale Beschreibungssprache zu entwickeln. Ein sich daraus ergebender Vorteil wäre neben der besseren maschinellen Bearbeitbarkeit eine Vereinheitlichung der Einträge und damit eine Standardisierung. Ein Formalismus um jeden Preis muß jedoch aus Gründen wie Unverständlichkeit und Unbeweglichkeit vermieden werden.

5.2.2 Erzeuger, Adressaten und Zugriffsberechtigungen

Eine letzte allgemeine Überlegung vor der konkreten Festlegung der Daten beziehe sich auf die Personengruppen, die Operationen mit diesen Daten durchführen. Besonders wichtig in diesem Zusammenhang ist die endgültige Klärung der Zugriffsrechte.

Drei Personengruppen haben das Recht und sind fachlich in der Lage, auf die Beschreibungsbank zuzugreifen:

- der Projektsekretär
- die Programmierer
- die mit der Wartung beauftragten Personen.

Betrachtet werden zunächst nur die ersten beiden, da die Wartung nicht zu den Produktionsphasen gehört, sondern zu einem späteren Zeitpunkt auf die Daten zugreift.

Der Projektsekretär in seiner Aufgabe als Koordinator ist zum größten Teil Datenkonsument. Sein System ist die Sammelstelle aller zu verwaltender Information und werde deshalb im folgenden als "Master-System" bezeichnet. Die bei einer Sammelaktion entstehenden Produkte sind Protokolle an den Projektleiter und aktualisierte Untersysteme für die einzelnen Programmierer, im folgenden "Slave-Systeme" genannt. Der Bearbeiter des Master-Systems muß die Berechtigung für lesenden und schreibenden Zugriff auf alle Daten besitzen. Nur mit diesem System dürfen für den allgemeinen Gebrauch nicht zugelassene spezielle Operationen wie etwa Änderung von Management-Information durchgeführt werden. Wegen ihrer zentralen Bedeutung und der damit verbundenen Sicherheitsanforderungen ist der Kreis der hier zugelassenen Bearbeiter auf den Projektsekretär und speziell von ihm beauftragte Personen einzuschränken. Dies kann und sollte durch ein paßwortähnliches Verfahren realisiert werden.

Die Programmierer sind die Erzeuger der meisten während des Produktionsverlaufs entstehenden Daten. Da jeder von ihnen an einer ihm zugewiesenen Teilaufgabe arbeitet, gibt es sicher definierte Datenbereiche, die sich dadurch auszeichnen, daß ein Programmierer exklusives Schreibrecht zu ihnen hat. Diese Datenbereiche beinhalten interne Aufgaben über einen Bereich, dessen Teile von ein und derselben Person bearbeitet werden. Während schon in Abschnitt 2 das Modul als kleinste logische Bearbeitungseinheit eingeführt wurde, besteht ein Bereich der jetzt auftretenden Art in der Regel aus mehreren Modulen; seine Einführung unterliegt nicht entwurfstechnischen sondern organisatorischen, arbeitseinteilenden Gesichtspunkten. Zu Beginn der Planungsphase ist seine spätere genaue Aufteilung in Module im Gegenteil noch nicht bekannt. Als ein Programmiererbereich soll im folgenden jeder Teil des zu produzierenden Systems bezeichnet werden, der einem Bearbeiter zur Realisierung und Dokumentation zugeteilt wurde.

Programmierer sollten auf die internen Daten nicht ihnen eigener Programmiererbereiche keinen schreibenden Zugriff besitzen. Dies ist eine Vorsichts-

maßnahme, die die nur in Extremfällen anzunehmende mutwillige, vor allem aber die fahrlässige und ungewollte Änderung nicht eigener Daten verhindern soll. Damit kann auch der Projektsekretär sicher sein, daß in einem Slave-System die programmierbereichinternen Daten höchstens im eigenen Bereich geändert sein können.

Die Grundlage zur Realisierung dieses Schutzes ist die maschinelle Kennzeichnung jedes Slave-Systems mit dem Namen seines Besitzers. Wird durch das System vor jeder Änderung interner Daten der Besitzername mit dem für den betroffenen Programmiererbereich eingetragenen Namen verglichen, so müßte im Extremfall ein Bearbeiter das Slave-System eines anderen an sich bringen, um in dessen Bereich Daten zu modifizieren. Der Datenschutz erfolgt also einfach durch Verschluß der Diskette mit dem eigenen Slave-System.

Für das Problem der Manipulation von Daten, die mehr als einen Programmiererbereich betreffen, also Schnittstellen beschreiben, bieten sich zwei Lösungen an:

- kraft seiner Zugriffsberechtigungen werden die Schnittstellendaten zentral vom Projektsekretär manipuliert
- nach einem festgelegten Protokoll führen die Programmierer diese Änderungen selbst durch.

Die Lösung 2 erweist sich als vorteilhafter. Das eingeführte Protokoll sieht vor, daß jeder Programmierer die vereinbarte Eintragung oder Änderung in seinem eigenen Bereich durchführt, das Beschreibungsbanksystem jedoch die Manipulation solange als Vorschlag verwaltet, bis ein neuer Sammellauf stattfindet. Bei der dann folgenden maschinellen Oberprüfung gibt es genau zwei Möglichkeiten:

- alle Programmierer, die die Schnittstelle betrifft, haben den gleichen Vorschlag eingetragen. Der Vorschlag wird akzeptiert und mit der Verteilung der neuen Slave-Systeme manifestiert.
- mindestens ein Programmierer hat keinen oder einen anderen Vorschlag eingetragen. Der Vorschlag wird ignoriert und der alte Zustand in allen neuen Slave-Systemen wiederhergestellt. Alternativ führt der Projektsekretär nach kurzfristiger Rücksprache mit den Betroffenen die Änderung durch.

Der Vorteil dieser Vorgehensweise liegt im erzwungenen Engagement der Programmierer. Sie müssen sich (mündlich) absprechen und einzeln die Änderung eintragen; war die Absprache eindeutig, so tritt sie in Kraft, ohne daß der Projektsekretär dies bemerkt. War die Absprache nicht eindeutig, so wird ihm der Fehler gemeldet.

Ein weiterer Vorteil der hier vorgeschlagenen Vorgehensweise liegt in der Überprüfung nur der notwendigen Schnittstellen, also derjenigen zwischen den Programmiererbereichen. Innerhalb seines eigenen Bereichs ist ein Programmierer völlig frei in seinem Modulaufbau und ihren Schnittstellen; die Probleme ergeben sich erfahrungsgemäß ja nur bei mehreren Bearbeitern.

Kurz werde die letzte Gruppe von Personen betrachtet, die mit der Beschreibungsbank arbeiten: die Wartungsingenieure. Sie sind bei ihrer Aufgabe, einen Fehler zu beseitigen, reine Datenkonsumenten, nach erfolgter Beseitigung Datenerzeuger, indem sie die Korrektur eintragen und das Wartungsarchiv führen. Da bei ihrem Einsatz das Projekt selbst bereits seinen Abschluß gefunden hat, existiert nur noch das Master-System. Dem Wartungsingenieur ist zu diesem System unbeschränkt Zugang zu gewähren, speziell sind ihm Operationen zur Verfügung zu stellen, die ein Archiv realisieren. Die oft unter dem Begriff Wartung geführten Systemerweiterungen sollten jedoch, sobald sie den Umfang größerer Korrekturen überschreiten, gesondert behandelt werden. Sobald sie den längeren Einsatz eines Programmierers oder eines Teams erfordern, sollte das Beschreibungsarchiv in seinen Produktionszustand zurückversetzt werden.

Bezogen auf die praktische Arbeit mit der Beschreibungsbank empfiehlt es sich, alle abzuspeichernden Daten mit einem Zugriffstyp zu versehen. Es gibt die Typen SYSTEM, SEMIPRIVATE und PRIVATE und fünf dazugehörende Regeln:

- alle Benutzer der Beschreibungsbank haben auf alle Daten freien lesenden Zugriff
- vom Master-System besteht freier schreibender Zugriff auf alle Daten
- der schreibende Zugriff auf Daten vom Typ SYSTEM ist von einem Slave-System nicht möglich
- der schreibende Zugriff auf Daten vom Typ SEMIPRIVATE von einem Slave-System ist möglich, wenn die Daten im Programmiererbereich seines Besitzers liegen. Die Modi-

fikation bedarf nach einem Protokoll der nachträglichen Zustimmung und kann zurückgewiesen werden

- der schreibende Zugriff auf Daten vom Typ PRIVATE von einem Slave-System unterliegt keiner Einschränkung, wenn die Daten im Programmierbereich seines Besitzers liegen.

5.2.3 Hierarchiemodell und Programmierbereich

Nach Abschnitt 2 ist es ein Grundprinzip des Software Engineering, bei der Planung eines Softwaresystems zu einem hierarchisch aufgebauten Entwurf zu gelangen. Die immer feiner aufgespaltenen Teilaufgaben, also nach dem Modell die einzelnen virtuellen Maschinen, sollen in ihrem Zusammenspiel einer Struktur gehorchen, deren Regeln genau festgelegt sind.

Ausgangspunkt der folgenden Betrachtungen sei die allgemeine Definition einer Relation zwischen zwei Knoten eines Systems. Ziel sei es, durch die Feststellung von Eigenschaften und durch die Einführung von Einschränkungen zu einer Beschreibung des im Rahmen dieser Arbeit verwendeten Modells eines hierarchischen Baums zu kommen.

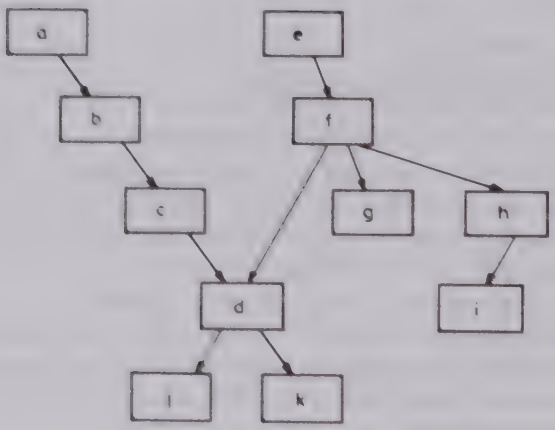
Die allgemeine Definition einer Relation (z.B. /PAR 74/) werde durch folgende Erklärung festgelegt:

> Eine zweistellige Relation R zwischen den Elementen der Menge M charakterisiert eine besondere Eigenschaft der geordneten Paare a, b ε M. Für jedes geordnete Paar muß eindeutig feststehen, ob es diese spezielle Eigenschaft innehat.

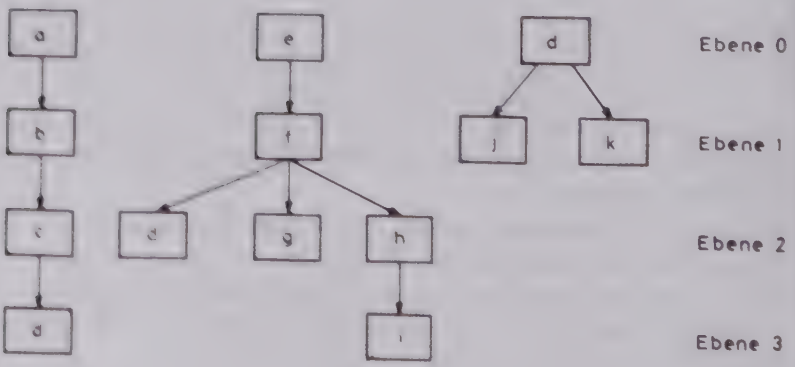
Die graphische Darstellung nach Bild 5.2.1a stellt die Elemente der Menge M als Knoten und die zwischen den Elementen existierenden Relationen als gerichtete Graphen dar.

Für den hier vorliegenden Anwendungsfall sind die Elemente der Menge M die Module eines Programmsystems. Die spezielle Eigenschaft der Relation R(a, b) zwischen den Modulen a und b hat die Bedeutung der Zugriffsmöglichkeit aus dem Modul a auf die nach außen definierte Schnittstelle des Moduls b.

a)



b)



c)

Bild 5.2.1 Zur hierarchischen Baumstruktur

Aus dieser Festlegung lassen sich sofort zwei spezielle Eigenschaften ableiten:

Die Relation R in der Bedeutung "hat Zugriff auf" genügt den Eigenschaften:

a) R ist asymmetrisch, d.h. für beliebige a, b ε M zieht die Gültigkeit von R(a, b) stets die Ungültigkeit von R(b, a) nach sich.

b) R ist irreflexiv, d.h. für ein beliebiges a ε M ist R(a, a) falsch.

Neben diesen allgemeinen Eigenschaften für die Relation R soll nun durch Definitionen die Struktur, also die Beziehungen zwischen den Elementen der Menge M in zwei Schritten geeignet eingeschränkt werden. Dazu werden zunächst formal Ebenen erklärt.

Zwischen den Elementen der Menge M werden Relationen R(a, b); a, b ε M eingeführt, die die Definition von Ebenen in folgender Weise ermöglichen:

a) Stufe $\emptyset$ ist die Menge von Elementen a, für die kein b mit R(b, a) existiert.

b) Stufe i, i ε $\mathbb{N}$ beinhaltet die Menge aller Elemente a, für die es auf Stufe i-1 ein b mit R(b, a) gibt.

c) Ist a ein Element auf Stufe i und gilt für irgendein Element c ε M die Relation R(c, a), so gehört c zu einer Stufe mit dem Index $\leq$ i-1.

Die Struktur eines nach diesen Regeln aufgebauten Systems heiße hierarchisch.

Bild 5.2.1b zeigt als Beispiel ein hierarchisch aufgebautes System, das auf Stufe $\emptyset$, 1, 3 und 4 je zwei und auf Stufe 2 drei Elemente beinhaltet. Um zu einem übersichtlich aufgebauten Programmsystem zu gelangen, soll noch eine weitere Einschränkung festgelegt werden:

Das aus der Menge M mit den Relationen R(a, b); a, b ε M aufgebaute System sei hierarchisch. Es werde definiert:

a) Es gibt genau ein Element auf Stufe $\emptyset$.

b) Zu jedem Element a auf Stufe i, i ε $\mathbb{N}$ existiert genau ein Element b auf Stufe i-1 mit R(b, a).

Die Struktur eines solchermaßen aufgebauten Systems heiße Baumstruktur.

Bild 5.2.1b stellt nach dieser Definition also keine Baumstruktur dar. Die dort gezeigte Struktur eines Programmsystems ist jedoch denkbar und auch übersichtlich. Aus diesem Grund erfolge eine letzte Festlegung:

> Seien M und S Mengen mit den Relationen R(a, b), die den Gesetzen einer Baumstruktur gehorchen. Sei weiterhin s das Element aus der Ebene Ø von S.
> S werde als Unterbaum (subtree) bezeichnet, wenn
>
> a) für jedes Element m ε M die Relation R(m, s) zugelassen ist,
>
> b) für alle Elemente x ε S die Relation R(x, z) falsch ist und z beliebige Elemente beliebiger Bäume darstellt.

Somit ist Bild 5.2.1b in Bild 5.2.1c auflösbar und genügt den Strukturregeln.

Die hierarchische Baumstruktur eines Programmsystems besteht nach diesen Definitionen im allgemeinen aus mehreren Bäumen Ø...n. Der Zugriff auf die Elementarfunktion, die ein spezieller Baum zur Verfügung stellt, kann nur durch Aufruf des Elementes in der Ebene Ø dieses Baums erfolgen.

Jedem der in einer Hierarchiestruktur existierenden Bäume muß einer der Typen ACTION oder SUBTREE zuweisbar sein. Die Unterstützung der Prozeßdatenverarbeitung erzwingt den weiteren Baumtyp INTERRUPT. Die Charakteristika der Baumtypen sind die folgenden.

Bäume vom Typ ACTION stellen in sich geschlossene Aufgaben im Sinne einer TASK in der Prozeßdatenverarbeitung dar. Ihr Ablauf kann von beliebiger Stelle durch später genau festzulegende Realtime-Anweisungen wie ACTIVATE, SUSPEND usw. beeinflußt werden.

Bäume vom Typ SUBTREE stellen in sich geschlossene Aufgaben im Sinne einer Prozedur dar. Ihre Aufrufmöglichkeit als sequentieller Ablauf aus beliebigen Ebenen beliebiger ACTIONS oder INTERRUPTs wird in der Hierarchiedarstellung durch den gerichteten Graphen angezeigt. Ist ein SUBTREE in einem speziellen Entwurf von mehr als einer ACTION aufrufbar, so ist bei seiner Realisierung die spezielle Eigenschaft seiner Reentrance-Fähigkeit zu berücksichtigen.

Bäume vom Typ INTERRUPT stellen in sich geschlossene Aufgaben dar, deren Ablauf nur durch ein asynchrones, externes Hardware-Ereignis aktiviert werden kann. Ihr Ablauf kann nicht durch Aufruf verlangt werden. Sie selbst jedoch können den Ablauf von ACTIONs durch Real-time-Operationen beeinflussen und sich mit ihnen über Datenbereiche synchronisieren.

Die Hierarchiestruktur eines echtzeitunabhängigen Softwaresystems besteht demnach aus genau einem Baum vom Typ ACTION, der Startpunkt, Gesamtaufgabe und Aufbau dokumentiert und beliebig vielen Bäumen vom Typ SUBTREE, die den Aufbau der auf mehreren Hierarchieebenen der ACTION verwendeten Unteraufgaben darstellen. Ein Echtzeitsystem besteht aus beliebig vielen Bäumen beliebigen Typs.

Zur Verwaltung der Hierarchiestruktur wurde in der Beschreibungsbank eine aus drei durch Punkte getrennte Dezimalzahlen bestehende Nomenklatur zur Bezeichnung jedes existierenden Knotens eingeführt. In der Reihenfolge ihres Auftretens bedeuten die Zahlen

- die laufende Nummer des Hierarchiebaums
- die Nummer der Hierarchieebene
- die laufende Nummer des Knotens in der Ebene.

Bild 5.2.1h zeigt an einer einfachen Struktur die Bezeichnungsweise in der Praxis. Dargestellt ist der Baum 7, er muß vom Typ ACTION oder INTERRUPT sein, denn er hat Zugriff auf einen Baum 9, der vom Typ SUBTREE sein muß.

Im abgeschlossenen Zustand, also bei Produktionsende, stellt die Hierarchiestruktur einen vollständigen Oberblick über das zu beschreibende Softwaresystem dar. Jeder Knoten steht für eine Teilaufgabe, die in sich geschlossen in einem eigenen Modul realisiert wurde.

Bei Produktionsbeginn jedoch ist der Kenntnisstand über den geplanten Systemaufbau natürlicherweise wesentlich gröber. Es sind Teilaufgaben mit definierten Schnittstellen festgelegt worden, und es liegt im allgemeinen fest, welcher

Programmierer welche dieser Teile realisieren soll. Es bietet sich also an, ausgehend von dieser groben Festlegung die Hierarchiestruktur mitlaufend zur Produktion verfeinern zu lassen. Dies geschieht in der Praxis nach folgendem Schema:

Zu Produktionsbeginn trägt der Projektsekretär am Master-System den bis dahin festgelegten groben Systemaufbau ein. Einem Knoten in der Hierarchiestruktur entspricht zu diesem Zeitpunkt eine in ihrer weiteren Aufspaltung noch nicht festgelegte Teilaufgabe. Nach dem organisatorischen Kriterium des gleichen Programmierers bilden jeweils einer oder mehrere miteinander verbundene Knoten einen Programmiererbereich. Im Laufe der Produktion verfeinert jeder Programmierer seinen Bereich zu der endgültigen Struktur. Jeder Sammellauf des Master-Systems übernimmt die durchgeführten Verfeinerungen, stellt eine neue Hierarchiestruktur auf und berechnet die neuen Knotenbezeichnungen.

Zur beliebigen Verfeinerung ihrer Bereichsstrukturen müssen den Programmierern die einfachen Grundoperationen nach Bild 5.2.2 zur Verfügung gestellt werden. Sie erlauben - innerhalb der festgelegeten Regeln - eine beliebige Strukturmodifikation.

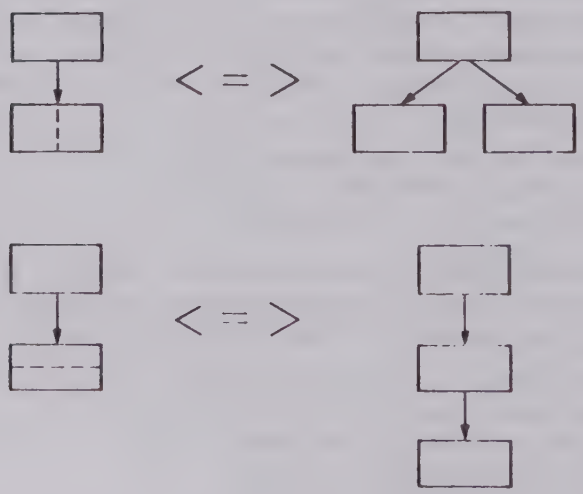


Bild 5.2.2 Grundoperationen zur Hierarchiestrukturmodifikation

Bild 5.2.3 zeigt für ein spezielles Beispiel die Grobstruktur bei Produktionsbeginn (a) und die endgültig verfeinerte Struktur nach erfolgter Produktion (b). Die Programmiererbereiche sind eingerahmt.

5.2.4 Die Beschreibung des Programmsystems

Die Oberlegungen des Abschnitts 5.2.1 zur Einteilung der abzuspeichernden Daten ergab eine Spaltung in einen allgemeinen und einen fachbezogenen Teil. Diese beiden Teile existieren sowohl für das zu beschreibende Programmsystem als Ganzes wie auch für jedes Modul. Betrachtet werde zunächst die Beschreibung des Programmsystems.

Eine das System als Ganzes betreffende Beschreibung hat die Aufgabe, den interessierten Personenkreisen einen schnellen umfassenden Oberblick zu verschaffen. Für die Interessenten der fachbezogenen Daten stellt die hierarchische Struktur diese Information dar. Sie sind mit ihrer Hilfe in der Lage, sich schnell über die Arbeitsaufspaltung innerhalb des Systems zu informieren und können bei Bedarf feststellen, welche genaueren Beschreibungen durchzuarbeiten sind. Was bisher fehlt sind die grundsätzlichen Kenndaten verwaltungstechnischer Art, die einen Oberblick über das Projekt geben. Die Beschreibungsbank bietet die Abspeicherung folgender Informationen:

- Programmsystemname, aktuelle Versionsnummer
  Jedes Programmsystem besitzt einen meist sehr einprägsamen kurzen Kunstnamen, der das Projekt identifiziert und über den das System oft später auch gestartet wird. Dieser Kunstname wird nicht selten mit werbungsähnlichem Geschick so gewählt, daß mit ihm selbst eine bestimmte Bedeutung verbindbar ist, er aber zusätzlich die Abkürzung einer näheren Systemerläuterung darstellt. Der Programmsystemname begleitet als Stichwort das gesamte Projekt und erscheint auf jeder Seite der Dokumentation. Ähnlich wichtig ist die Versionsnummer des Systems, die anzeigt, welche Systemversion in der Beschreibungsbank verwaltet wird. Der Zugriff auf die Beschreibung alter Versionen verlangt den expliziten Aufruf der Archivoperationen.

- Bearbeitungsfirma, Projektleiter, Auftraggeber, Obergabedatum
  Die Daten dieses Blocks zeigen dem ersten Leser, wer das Projekt durchführt und für wen die Durchführung stattfindet. Hier kann zum

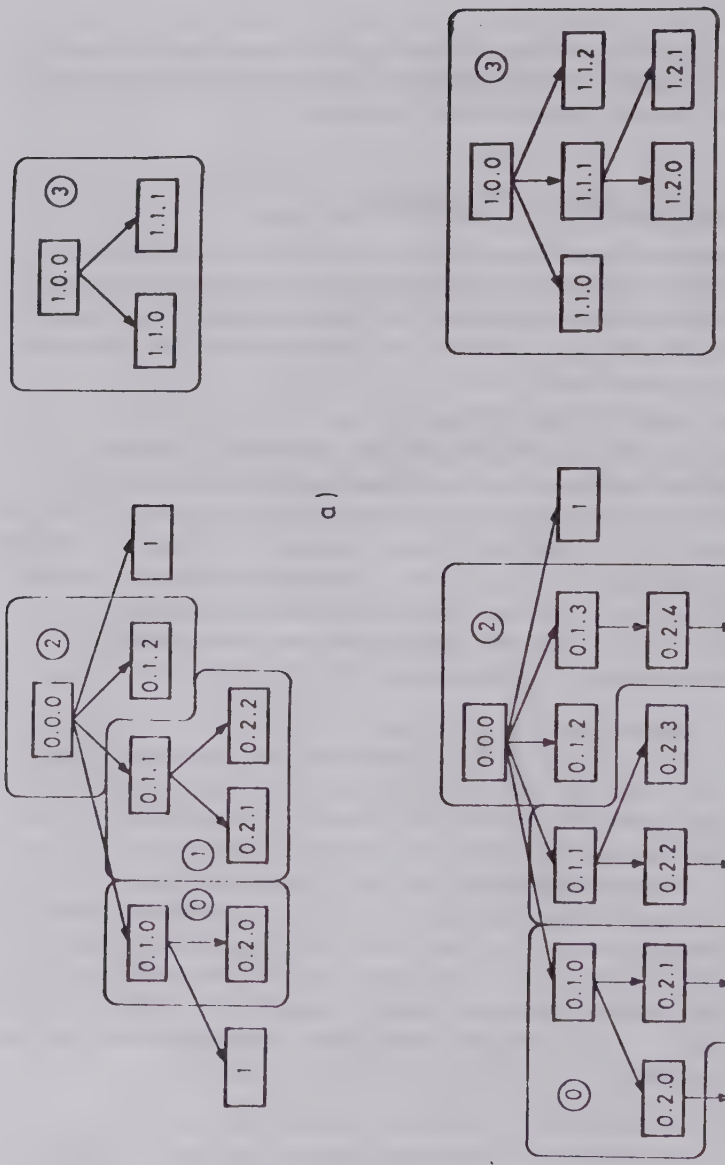


Bild 5.2.3 Hierarchiestruktur und Programmiererbereiche zu Produktionsbeginn und nach endgültiger Verfeinerung, Beispiel

Beispiel die Adresse einer Softwarefirma eingetragen werden, die für eine Industriefirma produziert; denkbar wären auch die Bezeichnungen zweier Abteilungen des gleichen Hauses. Das geplante Ende der Produktion als wichtigster Termin wird hier ebenfalls zentral gehalten.

- Rechnertyp, Betriebssystem, Hardwarevoraussetzungen
  In Stichworten werden in diesem Bereich Angaben zum Zielrechner gemacht. Dazu gehört die Rechnerbezeichnung, das darauf laufende Betriebssystem und Angaben über den vorausgesetzten Ausbau und die Konfiguration.

- Systemkurzbeschreibung, Systemhinweise
  Eine nicht mehr als etwa 20 Zeilen lange Beschreibung der Aufgaben, die das System aus Benutzersicht durchführt, soll auch dem nichtfachlichen Leser einen ersten Eindruck verschaffen können. Aus dieser Sicht stellt die Systemkurzbeschreibung eine äußerst kompakte Zusammenfassung der Spezifikation dar. Die Rubrik Systemhinweise wurde eingerichtet für Information weniger einordbarer Art, sie ist etwa nutzbar für Systemrestriktionen, später geplante Systemerweiterungen, besondere Installationsbedingungen, spezielle Vereinbarungen u.v.m..

- Literaturverweise, Kreuzverweise
  Eine letzte Rubrik der allgemeinen Systemdaten ist gedacht als Zeiger auf weiterführende Information, die nicht in der Beschreibungsbank enthalten ist. Dabei handelt es sich in der Regel um Titel und Standortverweise von Büchern, Ordnern oder anders publizierter Literatur, die in irgendeiner Form mit dem Projekt in Zusammenhang steht. Als Beispiel sei ein Verweis auf das Benutzerhandbuch oder auf ein Buch mit mathematischen Grundlagen zum Projekt genannt.

Alle genannten Daten, die die allgemeine Systembeschreibung darstellen, sind vom Typ SYSTEM. In einer schriftlichen Gesamtdokumentation des Systems erscheinen sie automatisch auf der ersten Seite nach einem Deckblatt, ansonsten ist ihre Protokollierung explizit einzeln aufrufbar.

5.2.5 Die Beschreibung der einzelnen Module

5.2.5.1 Kenndaten allgemeiner Art

Ähnlich dem Aufbau der Systemkenndaten sind auch die Beschreibungen der Einzelmodule nach Daten allgemeiner und fachlicher Art geordnet. Betrachtet werde zunächst der allgemeine Teil. Mit Rücksicht auf die Konsumenten dieser Daten werden von der Beschreibungsbank verwaltet:

- Modulname, aktuelle Versionsnummer
  Ziel dieser Grunddaten ist - wie beim System selbst - die eindeutige Identifizierung der folgenden Information. Der Modulname und gegebenenfalls die Versionsnummer dienen also als Schlüssel zum Zugriff auf alle dieses Modul betreffende Beschreibungen. Daraus resultiert die Forderung, daß der gleiche Modulname nicht mehrmals im System vorkommen darf. Nur eine Version eines Moduls - in der Regel die aktuellste - wird bei Aufruf automatisch bearbeitet; auf die Beschreibungen anderer Versionen muß über die Archivoperationen vom Master-System zugegriffen werden.

- Knotentyp, Knotennummer
  Der Knotentyp gibt darüber Auskunft, welche Stellung das Modul in der Hierarchiestruktur innehat. Er wird formal angegeben und kann die Zustände ACTION, SUBTREE, INTERRUPT, MODULE und DATA annehmen. Die ersten drei dieser Begriffe wurden bereits in Abschnitt 5.2.3 definiert; einen dieser drei Zustände bekommt jeweils der Knoten in der Ebene Ø jedes Baums zugewiesen. Alle anderen zu Bäumen gehörende Knoten sind vom Typ MODULE. Eine Sonderstellung nimmt der Typ DATA ein. Er dient, wenn nötig, zur Beschreibung spezieller Einheiten, die nur aus Datendefinitionen bestehen, etwa einem BLOCKDATA in FORTRAN oder einem PL/1-Modul, daß nur Datendeklarationen (mit dem Attribut PUBLIC) enthält. Die Beschreibung eines Knotens vom Typ DATA enthält naturgemäß keine Angaben über ihren Programmablauf. Der Knoten ist keinem Baum angegliedert, sondern nimmt eine Sonderstellung in der Hierarchie ein.

  Die Knotennummer wird einem Modul von der Beschreibungsbank selbst zugewiesen und stellt eine Verbindung zwischen Hierarchiestruktur und Beschreibungen dar.

- Programmiersprache
  Mit Rücksicht auf die nicht abwegige Tatsache, daß verschiedene Teile eines Programmsystems in verschiedenen Programmiersprachen implementiert sein können, wird getrennt für jedes Modul der Name der gewählten Programmiersprache verwaltet. Diese Angabe dient zunächst nur der Dokumentation, es wird sich jedoch im folgenden zeigen, daß sie auch Grundlage weiterer maschineller Tätigkeiten sein kann.

- Bearbeiter, Bearbeitungszustand
  Dies sind nun eindeutig Managementinformationen. Die Namen der Bearbeiter bestimmen die nach organisatorischen Gesichtspunkten festgelegten Programmiererbereiche und dienen zusammen mit der internen Angabe über den Besitzer des Slave-Systems zur Oberprüfung der schreibenden Zugriffsberechtigungen.

  Der Bearbeitungszustand jedes Moduls hält im Laufe der Produktion seinen Werdegang fest und beschreibt zu jeder Zeit seinen aktuellen Zustand. Es gibt formal die Zustände GEPLANT, ENTWORFEN, CODIERT, GETESTET, 1. FREIGABE, 2. FREIGABE; jeweils ist der Eintrag eines Solldatums und eines Istdatums möglich. Der letzte Zustand kommt nur dann zum Tragen, wenn das zum ersten Mal freigegebene Modul fehlerbehaftet war, so daß es an den Bearbeiter zurückgewiesen werden mußte.

- Modulkurzbeschreibung, Modulhinweise, Kreuzverweise
  Analog zu den entsprechenden Rubriken der allgemeinen Systembeschreibung dienen die hier aufgeführten Angaben einer kurzen übersichtshaften Beschreibung der Modulaufgabe, weiterer freier Angaben zum Modul und möglicher Verweise auf weiterführende Literatur.

- Letzte Änderung: Datum, Bearbeiter, Grund
  Zur Führung des Archivs sind bei Erstellung einer neuen Modulversion als Minimum die obengenannten Maintenance-Angaben nötig. Nur so ist ein lückenloses Bild der Modulgeschichte garantiert. Diese Angaben sind unabhängig von der eventuellen Archivierung oder Löschung einer Version selbst; die Löschung einer sowieso fehlerhaften Version kann sinnvoll sein, die nicht erfolgte oder gelöschte Angabe der Maintenance-Daten ist immer unsinnig.

- Versionsarchiv: Versionsnummern, Datenträgerangaben
  Zur Vervollständigung des mit den Änderungsangaben begonnenen Archivs dienen die letzten Daten der allgemeinen Modulbeschreibung als Inhaltsverzeichnis aller existierenden Versionen dieses Moduls. Für jede Version werden ihre Nummer und - wenn noch vorhanden - der Dateiname und das Label des Datenträgers verwaltet, auf dem die Quelle, das Objekt und die alte Beschreibung "eingemottet" wurde.

Alle genannten Daten bis auf die im folgenden genannten Ausnahmen sind vom Typ SYSTEM und werden bei der Festlegung der Programmiererbereiche vom Projektsekretär eingetragen. Bei einer vom Programmierer durchgeführten Grundoperation zur Knotenspaltung (Bild 5.2.2) werden alle allgemeinen Daten vom Typ SYSTEM bis auf den Modulnamen übernommen.

Die Angaben zur Modulkurzbeschreibung haben den Datentyp PRIVATE. Die Angabe der Programmiersprache und die Ist-Daten des Bearbeitungszustandes werden SEMIPRIVATE behandelt. Hier durchgeführte Eintragungen und Änderungen bedürfen der Zustimmung des Projektsekretärs am Master-System.

5.2.5.2 Beschreibung der Modulschnittstellen

Die von der Beschreibungsbank vorgeschriebene Struktur des Systemaufbaus sieht mit dem Hintergrund der Obersichtlichkeit eine strenge Einschränkung möglicher Aufrufe vor. Ein MODULE darf nur aus der nächsthöheren Hierarchieebene nach Art einer Prozedur, also im Sinne der Anweisung CALL, aufgerufen werden. Ihre Aufgabe wird sequentiell abgearbeitet und die Kontrolle zum Aufrufenden zurückgegeben. Nicht ganz so streng ist die Beschränkung für einen SUBTREE. Er darf prozedurartig aus allen Modulen eines Baums vom Typ ACTION oder INTERRUPT erreicht werden.

Komplex gestaltet sich der Aufruf des Typs ACTION. Sein Aufruf ist aus allen Modulen erlaubt, seine Aufrufform und sein Ablauf jedoch sind an spezielle Anweisungen gebunden, die den prioritäts- und echzeitabhängigen Betrieb paralleler Prozesse unterstützen. Zur zeichnerischen Darstellung von Ablaufstrukturen dieser Art sind die bisher verwandten Hierarchiestrukturen ungeeignet. Diese Darstellungen werden vielmehr vorteilhaft etwa mit Hilfe von Petri-Netzen durchgeführt.

Bäume vom Typ INTERRUPT und natürlich DATA sind nicht aus der Struktur aufrufbar.

Die enge Handhabung der Zugriffsberechtigungen gilt jedoch in der Beschreibungsbank nicht für gemeinsame Datenbereiche. Für einfache und zusammengesetzte Daten beliebigen Typs kann ein Geltungsbereich auf das Gesamtsystem, auf beliebige Einzelmodule oder auf das Innere eines Moduls festgelegt werden. So ist es sinnvoll, für jedes Modul diejenigen Daten und Datenbereiche zu dokumentieren, die es mit anderen Modulen verbindet.

Nach diesen Vorüberlegungen sind zur vollständigen Dokumentierung aller Schnittstellen eines Moduls mit seiner Umgebung die folgenden Informationen ausreichend:

- der Name, über den das Modul aufrufbar ist und die Form eventuell dabei übergebener Daten (formale Parameter)
- die Namen, über die das Modul andere Module nach Art einer Prozedur aufrufen kann und die Form der eventuell dabei übergebenen Daten
- die Namen aller ACTIONs, die das Modul mittels Real-Time-Anweisungen beeinflussen kann
- Die Namen und die Form aller Daten und Dateien, deren Geltungsbereich sich nicht auf das Innere des Moduls beschränkt.

Diese Zusammenfassung bedarf einiger Erläuterungen und genauerer Ausführungen.

Der Name, über den die Aufgabe eines Moduls aufrufbar ist, ist im allgemeinen der Name der in diesem Modul enthaltenen von außen erreichbaren Prozedur. Es kann programmiersprachenabhängig vorkommen, daß dieser Prozedurname identisch mit dem Modulnamen ist. Die damit implizit gemachte Einschränkung, daß ein Modul nur eine von außen erreichbare Prozedur enthalten darf, wird von vielen Autoren und auch von der Syntax vieler Programmiersprachen nicht unterstützt. Die hier eingeführte strenge Reglementierung geschieht aus Gründen der Oberprüfbarkeit. Da jedes Modul genau eine Teilaufgabe enthalten soll, erscheint es folgerichtig auch sinnvoll, den Aufruf dieser Aufgabe nur unter einem Namen zuzulassen und während ihres Ablaufs genaue Kenntnis über die unter diesem Namen bekannten Datenbereiche zu haben.

Aktuelle Daten können mit einem aufgerufenen Modul auf zwei Arten getauscht werden: sie werden bei der Übergabe der Programmkontrolle übergeben, oder sie stehen in einem dem aufrufenden und dem aufgerufenen Modul zur Verfügung stehenden Datenbereich.

Alle Programmiersprachen, die die Übersetzung einzelner Module erlauben, kennen die Attribute PUBLIC (oft auch GLOBAL o.ä.) und EXTERNAL (auch EXTRN o.ä.). Diese sowohl auf die Aufrufnamen als auch auf die Daten anwendbaren Attribute geben deren Herkunft an und sollen hier übernommen werden. In der Schnittstellenbeschreibung vorkommende Namen mit dem Attribut PLC sind im Inneren des beschriebenen Moduls definiert und werden nach außen zur Verfügung gestellt. Namen mit dem Attribut EXT sind in einem anderen Modul definiert und im beschriebenen Modul wird auf sie zugegriffen.

Die Namen aller in einem Modul beeinflußten ACTIONs dienen zusammen mit den in der Ablaufbeschreibung dokumentierten auf sie angewandten Anweisungen als Grundlage für geplante maschinelle Verfahren zur Simulation des Echtzeitverhaltens und zur Kontrolle möglicher Ablaufverklemmungen.

Es ist sinnvoll, die Form der dokumentierten Datenbereiche programmiersprachenunabhängig anzugeben. Insbesondere muß im Hinblick auf die dadurch entstehenden Dokumentationsvorteile neben der Bereitstellung von Standarddatentypen eine Möglichkeit bereitgestellt werden, eigene Datentypen zu definieren und zu verwenden. Weiterhin muß die Möglichkeit bestehen, zusammengesetzte Datentypen zu dokumenteren.

Standardmäßig stellt die Beschreibungsbank folgende Datentypen zur Verfügung:

INTEGER	beschreibt eine ganze, positive oder negative Zahl
REAL	beschreibt eine reelle Zahl in Gleitpunktdarstellung
LOGICAL	beschreibt die beiden möglichen logischen Zustände TRUE und FALSE
CHAR	beschreibt alle darstellbaren Zeichen
STRING	beschreibt eine Kette aus darstellbaren Zeichen
POINTER	beschreibt einen nur als Datenzeiger verwendbaren Wert
SEMA	beschreibt einen zur Koordinierung von ACTIONs verwendbaren Wert.

Weitere eigene Datentypen mit ihren erlaubten Zuständen können - für das gesamte System verbindlich - durch einen speziellen Befehl vom Master-System definiert werden.

Jedem in die Beschreibungsbank eingetragenen Variablennamen muß neben seinem Attribut (EXT oder PLC) ein erlaubter Datentyp zugewiesen werden. Weitere optionale Angaben, die sich bei gleichen Namen von Modul zu Modul ändern können, umfassen die Erklärung eines erlaubten Wertebereichs als Untermenge des mit dem Datentyp festgelegten Umfangs und eine Zugriffsberechtigung (lesend, schreibend).

Die Wiederholung von Daten, also die Datenform des ARRAY, ist in der Beschreibungsbank darstellbar, indem die Anzahl der Wiederholungen in runden Klammern auf den Namen folgend eingetragen wird. Es sind bis zu sieben Dimensionen erlaubt, die einzelnen Dimensionsgrößen sind dann durch Komma zu trennen. Die Darstellung einer Datenstruktur, also die unter einem Namen zusammengefaßte Menge logisch zusammengehörender Einzeldaten, ist mit dem RECORD möglich. RECORDs dürfen auch geschachtelt auftreten, jedoch sollte bei deren Verwendung berücksichtigt werden, daß sich bei Programmiersprachen, deren Syntax diese Konstruktionen nicht unterstützt, eine allzu starke Abstraktion der Dokumentation von der Realisierung ergeben kann.

Schließlich ist es bei der Datenbeschreibung möglich, jedem Namen einen Kommentar zuzuordnen; eine weitere Unterstützung, falls die Datenbeschreibung selbst nicht deutlich genug erscheint.

Bild 5.2.4 zeigt in einem speziellen Beispiel die Anwendung der Datenbeschreibung. Beschrieben wird eine Datenstruktur, die bestimmte Kenndaten für 20 Häuser mit je 10 Etagen aufnehmen soll. Voraussetzung zu dieser Beschreibung ist die irgendwann vorher erfolgte Definition eines eigenen Datentyps, der hier verwendet wird. Dies könnte am Master-System etwa in der Form

DATENTYP, CAR = (RENAULT, VW, AUDI, OPEL, FORD, TOYOTA)

erfolgt sein.

Der Beschreibung der in einem Modul bearbeiteten Dateien dient ein eigener Abschnitt der Beschreibungsbank. Dabei wird kein logischer Unterschied zwischen physikalisch auf externen Massendatenspeichern (Platte, Band) abgelegten Dateien und durch periphere E/A-Geräte realisierten Dateien gemacht. Eine solche Unterscheidung erscheint schon deshalb unsinnig, da in vielen Programmiersprachen die Ein- und Ausgabe über logische Device-Nummern ab-

Name	Typ	Zugriff	Bereich	Kommentar
Wohnblock (20)	RECORD			Wohnblockbeschreibung
Strasse	STRING	L		Adresse
Hausnummer	INTEGER	L	1...200	
Etage (10)	RECORD			Mieterdaten
Mietername	STRING	L		
Vorname$Abk	CHAR	L	A...Z	
Verheiratet	LOGICAL	L		
Automarke	CHAR	S		
Hausmeister	STRING	L		

Bild 5.2.4 Beispiel für die Dokumentation einer Datenstruktur

gewickelt wird und für jeden Lauf erst bei Programmstart eine Ankopplung an spezielle Dateien erfolgt. Die damit verbundenen Gefahren unsachgemäßer Zugriffe (z.B. der Versuch, von einem Schnelldrucker zu lesen) lassen sich prinzipiell nicht zur Dokumentationszeit vermeiden.

Zur möglichst vollständigen Dokumentation wird für jedes Modul gespeichert:

- Der Name jeder bearbeiteten Datei. Zur flexiblen Handhabung läßt die Beschreibungsbank sowohl Device-Nummern wie auch explizite Dateinamen zu. Da die Syntax von Dateinamen jedoch stark vom eingesetzten Betriebssystem abhängt, ist eine allgemeine weitere Überprüfung hier kaum möglich.
- Der Typ jeder bearbeiteten Datei. Im Rahmen der Beschreibungsbank wurden in einer ersten Ausbaustufe zunächst nur drei Dateigrundtypen berücksichtigt. Danach hat jede Datei alternativ den Typ:

  SEQ die rein sequentiell aufeinanderfolgende Lagerung von Nutzdaten

  ISEQ die sequentielle Lagerung von Nutzdaten und ihre Verwaltung über eine oft mehrstufige Indexliste

DIR die direkte Lagerung von Nutzdaten mit wahlfreiem Zugriff über eine Abbildungsfunktion (z.B. ein Hash-Code).

- Die mit jeder Datei in diesem Modul durchgeführten Aktionen. Mögliche auf Dateien anwendbare Aktionen lassen sich auf recht wenige Grundoperationen (z.B. OPEN, READ, WRITE, CLOSE) zurückführen, die in der Regel ein oder mehrere Betriebssystemaufrufe darstellen. Eine Liste aller für die Beschreibungsbank standardisierten auf eine Datei anwendbaren Aktionen ist im Anhang zu finden.
- Ein beliebig langer Kommentar zu jeder beschriebenen Datei.

5.2.5.3 Beschreibung des Modulablaufs

Um die Beschreibung eines Moduls zu komplettieren, fehlt nun noch eine Unterstützung zur Dokumentation seines Ablaufs. Dokumentationsangaben dieser Art unterliegen mehreren Anforderungen. Zunächst ist es, genau wie bei den Modulschnittstellen, sinnvoll, die Beschreibung programmiersprachenunabhängig durchzuführen. Die Erfüllung dieses Kriteriums ist eine notwendige Voraussetzung für eine weiterhin zu fordernde Dokumentationsart, die die logischen Ablaufstrukturen hervorhebt, ohne einen zu großen Detaillierungsgrad zu besitzen. Nur so besteht die Chance, daß ein Durcharbeiten der Dokumentation zum Verständnis des Modulaufbaus beiträgt. Dagegen führt das naturgetreue Abbild jedes Programmstatements in den Dokumentationsunterlagen mit Sicherheit zu ihrer Ignorierung, weil sich dann die eigentlich oft unübersichtliche Programmquelle zur Einarbeitung doch als erfahrungsgemäß effektiver erweist.

Alle Dokumentationen, die eine zeichnerische Darstellung unbedingt voraussetzen, sind abzulehnen. Steht kein maschinelles Zeichengerät, etwa ein Plotter, zur Verfügung - und die Konfiguration der Beschreibungsbank sah kein solches vor - ist die Erstellung stilisierter Graphiken nur unter erschwerten Bedingungen möglich und das Ergebnis nicht zufriedenstellend. So seien zum Beispiel Programmpakete genannt, die eine Dokumentation in Form von Programmablaufplänen erzeugen, deren Symbole aus druckbaren Zeichen zusammengesetzt und auf einem Schnelldrucker ausgebbar sind. Die unübersichtlichen Ergebnisse rechtfertigen in keinem Fall den sicherlich hohen Aufwand ihrer Erzeugung.

Eine weitere Anforderung an die Dokumentation von Ablaufstrukturen ergibt sich aus der allgemeinen Beschreibung der Phasen des Software Engineering

(Abschnitt 2). Dort wurden die Vorteile zusammengestellt, die mit dem Aufbau von Ablaufstrukturen nach den Regeln der strukturierten Programmierung verbunden sind. Darauf aufbauend sollte eine Dokumentation solcherart entworfener Programme die entstandenen Strukturblöcke automatisch hervorheben und ihre Schachtelung sichtbar machen.

Schließlich gilt es, die schon allgemein genannte Forderung nach einheitlicher, formaler Dokumentation soweit wie möglich zu berücksichtigen.

Die im Beschreibungsbanksystem festgelegte Methode zur Dokumentation von Ablaufstrukturen ist halbformaler Natur. Sie besteht zunächst aus beliebig viel Umgangssprachentext, der formatfrei eingetragen und unverändert übernommen wird. Bestimmte Zeichenkombinationen werden jedoch erkannt und dienen einer maschinellen Aufarbeitung und Analyse. Diese flexible Vorgehensweise erlaubt eine beliebige verbale Dokumentation, die auch dann akzeptiert wird, wenn die Beschreibungsbank keine ihr bekannten speziellen Zeichenkombinationen findet. Der eigentliche Vorteil liegt jedoch in ebenderen Verwendung. Folgende Strings werden gesondert bearbeitet:

- Modul-, Daten- und Prozedurnamen

  Alle der Beschreibungsbank als Modul-, Daten- oder Prozedurnamen bekannten Strings werden aus dem verbalen Text gefiltert, besonders gekennzeichnet und mit der Hierarchienummer des Moduls versehen, in dem sie definiert wurden. Sie werden als unzulässig erkannt, wenn sie nicht explizit in der Schnittstellenbeschreibung aufgeführt sind.

- Schlüsselworte, die Ablaufstrukturen kennzeichnen

  Zu einer speziellen Bearbeitung führt das Erkennen fest definierter Schlüsselworte, mit denen der Erzeuger der Dokumentation die Verwendung von Strukturblöcken kennzeichnen kann. Eine so beschriebene Ablaufstruktur kann vom System auf die Regeln der strukturierten Programmierung überprüft werden. Eine als erlaubt erkannte Struktur kann vom System optisch aufbereitet und so in übersichtlicher Form dargestellt werden.

- Real-time-Statements

  Das Erkennen eines Elements aus einer festgelegten Liste von Real-time-Statements führt zu einer Oberprüfung ihrer gültigen

Anwendung auf einen direkt danach erwarteten ACTION-Namen. Ebenso wird ein auftretender ACTION-Name ohne zulässiges Aufruf-Statement als Fehler erkannt.

Die beschriebenen, im System als spezielle Zeichenkombinationen vereinbarten Strings sollten nach Möglichkeit nur in der damit erwarteten Bedeutung verwendet werden. Da jedoch Möglichkeiten denkbar sind, die die Verwendung dieser Strings sinnvoll erscheinen lassen, ohne daß ihre spezielle Identifizierung durch das System erwünscht ist, kann diese Identifizierung an beliebiger Stelle durch Eingabe eines Sonderzeichens explizit verhindert werden. Ebenso ist es möglich, die Abspeicherung einer als fehlerhaft erkannten Dokumentation zu verlangen; sie wird durch das System als fehlerhaft gekennzeichnet und kann zu einem späteren Zeitpunkt korrigiert werden.

Grundsatz des soeben verbal beschriebenen Verfahrens ist die Programmablaufdokumentation durch eine sinnvolle Mischung aus Schlüsselwörtern und Umgangssprachentext. Die damit verbundenen Vorteile werden jedoch nur dann vollständig ausgenutzt, wenn die darin integrierte Fähigkeit der Überprüfung von Strukturblöcken ausgenutzt wird. Die Akzeptanz einer solcherart aufgebauten Dokumentation ist damit zwar sicher kein hinreichendes, aber bestimmt ein notwendiges Kriterium für einen übersichtlichen, durchschaubaren und leicht modifizierbaren Feinentwurf eines Moduls. Voraussetzung dafür ist jedoch ein formal genau festgelegter Satz von Regeln, die spezifizieren, welche Strukturen das System akzeptiert.

Zur schriftlichen Niederlegung dieser Regeln bietet sich ihre Beschreibung in einer Metasprache in der von Backus und Naur /NAU60/ vorgeschlagenen Form an. Diese nach Art von Produktionen formulierte Darstellungsart bietet mehrere Vorteile. Zum einen ist der Regelsatz nach kurzer Einarbeitung jedem Programmierer verständlich, zumal oft auch die Grammatik von Programmiersprachen in dieser Form beschrieben wird, zum anderen begünstigt die formale Art ihre Implementierung auf einem Rechner.

In einer ersten Implementierung bestehen die die erlaubten Strukturen beschreibenden Regeln aus 11 Produktionen. Die damit festgelegte einfache Grammatik lautet:

```
<modul>              ::= {<prozedur>}_0^∞ <block> END
<prozedur>           ::= <name> : PROCEDURE <block> END
<block>              ::= <text> | {<text>} <elementarstruktur> {<text>}
<elementarstruktur>  ::= <if-struktur> | <while-struktur> | <until-struktur>
                         | <for-struktur> | <switch-struktur>
<if-struktur>        ::= IF <text> THEN <block> {ELSE <block>} ENDIF
<while-struktur>     ::= WHILE <text> DO <block> ENDW
<until-struktur>     ::= REPEAT <block> UNTIL <text> ENDR
<for-struktur>       ::= FOR <text> TO <text> {STEP <text>} DO <block> ENDF
<switch-struktur>    ::= SWITCH <text> {CASE <text> DO <block>}_2^∞
                         {DEFAULT <block>} ENDC
<name>               ::= <buchstabe> {<buchstabe> | <ziffer>}_0^11
<text>               ::= <name> | <bel. Zeichenfolge, die keine Namen
                         und keine Schlüsselworte enthält>
```

Mit dieser Festlegung sind alle Daten beschrieben, die die Beschreibungsbank in ihrer jetzigen Form zur Verfolgung und zur Dokumentation eines Softwareprojekts aufnehmen kann. Sie erlauben eine vollständige Beschreibung nahezu jedes Softwareprojekts, dessen Beschreibungsdatenmenge den vorausgesetzten physikalischen Massenspeicherrahmen nicht überschreitet.

Eine komfortable Arbeitsweise des Benutzers mit der Beschreibungsbank durch ein geeignetes Dialogsystem erlaubt nun neben reinen Grunddiensten wie Eintragen und Protokollieren komfortable auf diese Daten angewandte auswertende und generierende Dienste. Die sich ergebenden Möglichkeiten soll anhand einer konkreten Realisierung der nächste Abschnitt zeigen.

"Why", said the Dodo,
"the best way to explain it is to do it."

Lewis Caroll: Alice in Wonderland

6. Die Realisierung einer Beschreibungsbank

Nach den ausführlichen Oberlegungen zu den Anforderungen an eine softwareproduktionsbegleitende Beschreibungsbank soll nun eine konkrete Realisierung beschrieben werden. Für dieses in Teilen realisierte und in Teilen geplante System sollen in Aspekten seine Umgebung, seine Bedienung und sein innerer Aufbau dargelegt werden.

6.1. Die Umgebung der Beschreibungsbank SCROLL

Im Rahmen dieser Arbeit entstand unter Berücksichtigung der bisher aufgezeigten Oberlegungen eine konkrete Beschreibungsbank, das System SCROLL (Software Control related on a leading Library). Im folgenden Abschnitt soll zunächst das Rechnersystem vorgestellt werden, auf dem SCROLL implementiert wurde. Ziel dieses kurzen Exkurses sei das Verständnis für die standardmäßig gebotene Unterstützung durch den Rechner, aber auch für die rechnerbedingten Einschränkungen und Grenzen. Implementierungssystem ist ein Mikrorechnerentwicklungssystem der Firma Intel Comp., USA. Die amerikanische Originalbezeichnung lautet MDS (Microcomputer Development System) Serie II, in der BRD wird ein baugleiches System von der Firma Siemens mit der Bezeichnung SME (Siemens Mikrocomputer Entwicklungssystem) vertrieben.

Kern der Anlage ist der schon recht alte und bewährte Intel-Baustein 8085, ein 8-Bit-Prozessor mit 16 Bit Adreßbus. Der Zentralspeicher ist voll ausgebaut mit 64K x 8 Bit, jedoch stehen, wie bei Besprechung der Systemsoftware näher erläutert wird, dem Anwender nur definierte Teilbereiche zur Verfügung. Die Konsol-E/A erfolgt über ein im Rechnergehäuse integriertes Sichtgerät und eine ASCII-Tastatur mit einigen Sonderzeichen. Der Monitor des Sichtgeräts wird gesteuert durch einen integrierten CRT-Controller vom Typ 8275, einem komfortablen Baustein, der unter anderem auch die blinkende und inverse Darstellung von Zeichen unterstützt. Diese gerade für die schon beschriebene Menü-Technik sehr interessante Eigenschaft wird jedoch offiziell nicht über Systemdienste zur Verfügung gestellt, vielmehr beschränkt sich die Unterstützung der Konsol-E/A in Schnittstellen für Einzelzeichentreiber.

Zum Anschluß weiterer Peripheriegeräte stehen parallele und serielle Schnittstellen zur Verfügung; standardmäßig ist ein Drucker für 132 Zeichen breites Papier mit einer Schreibgeschwindigkeit von 200 Zeichen/sec angeschlossen. Weiterhin ist ein intelligenter DIN-A-3-Plotter vorhanden, der über eine serielle Schnittstelle betreibbar ist und die Erstellung von Zeichnungen durch einen komfortablen Befehlssatz unterstützt.

Als Massenspeicher stehen Magnetspeicher in Floppy-Disk-Technologie zur Verfügung, maximal sind vier Laufwerke betreibbar; der aktuelle Ausbau besteht aus zwei Laufwerken, single-sided mit Double-Density-Aufzeichnung. Damit ist aktuell auf etwa 1 Megabyte Rohdaten ohne Datenträgerwechsel zugreifbar.

Vom Hersteller angeboten wird auch ein kombiniertes Fest- und Wechselplattenlaufwerk, damit erhöht sich die Massenspeicherkapazität auf etwa 7.6 Megabyte. Ein unabhängiger Hersteller bietet ein hardware- und softwarekompatibles Laufwerk in Winchester-Technologie mit einer Kapazität von ca. 29 Megabyte (unformatiert) an.

Die Systemsoftwareunterstützung besteht zunächst aus einem residenten Betriebssystemteil, der bei Rechnerinitialisierung von Floppy-Disk oder Platte geladen wird und im folgenden über eine definierte Systemeinsprungstelle zur Verfügung steht. Seine Größe beträgt etwa 13K byte, seine Leistungen umfassen eine kombinierte Massenspeicherdateiverwaltung und eine Peripheriegerätesteuerung sowie einen Lader, der Programme vom Massenspeicher in den Zentralspeicher transportieren und starten kann. Weiterhin steht ein 2K byte großer Monitor in Festwertspeicher zur Verfügung, er dient jedoch ausschließlich zu Testzwecken. Alle weiteren Systemdienstprogramme (Editor, Compiler, ...) und alle Anwenderprogramme werden bei Bedarf in den nun noch freien Zentralspeicherbereich geladen und ausgeführt. Auch eine Segmentierung sehr großer Programme in Overlaybereiche ist möglich, wird jedoch vom System in ihrer Erstellung, etwa durch Angabe von Overlaybereichen an den Binder, nicht unterstützt.

Die einzig mögliche und durch den Filemanager über Systemaufrufe bearbeitbare Dateiform ist die einfache sequentielle Datei. Die wichtigste Systemunterstützung besteht aus dem Öffnen einer Datei (OPEN, impliziert ihre Kreation), ihrem Schließen (CLOSE), ihrem Löschen (DELETE) und den sequentiellen Lese- und Schreiboperationen (READ und WRITE). Eine interessante weitere Eigenschaft ist die zusätzliche Möglichkeit, in der Datei vorwärts und rück-

wärts zu positionieren (SEEK). Damit läßt sich mit einigem eigenen Softwareaufwand (Listenverwaltung) eine indexsequentielle Dateiform simulieren, eine für eine datenbankähnliche Anwendung unabdingbare Voraussetzung.

Die Sprachunterstützung des Systems besteht aus einem Assembler für 8085-Code (/INT 77a/), einem PL/M-Compiler (/INT 77b/) und einem FORTRAN-Compiler (/INT 78a/). Interpreter stehen für die Sprachen BASIC (/INT 78b/) und PASCAL (/INT 79/) zur Verfügung. Jedoch ist der Anwender bei der Auswahl der für sein Anwenderprogramm geeigneten Programmiersprache trotz dieser relativen Vielfalt recht eingeschränkt. Sprachinterpreter sind für die meisten Systeme, speziell jedoch für Dialogsysteme, aufgrund ihrer Ausführungszeiten abzulehnen. Die Assemblersprache eignet sich sicher bei Betrachtung der Rechnerausnutzung (Codemenge, Laufzeit) optimal, ist jedoch in den meisten Fällen aufgrund des mangelnden Programmierkomforts und der langen Systemerstellungszeit abzulehnen. Doch selbst aus FORTRAN- oder PL/M-Quellen compilierte Programme zeigen sich im Vergleich zu ihrer Realisierung in Assembler ausgesprochen uneffektiv. So ergaben Messungen des Autors (/GEO 77/) bei der gegebenen Realisierung von Aufgaben in Assemblersprache eine Erhöhung von Codemenge und Laufzeit von ca. 100% bei ihrer Realisierung in PL/M, für FORTRAN ergaben sich Faktoren von 4...6. In Zusammenfassung dieser Faktoren kommt, bei einem Mittelweg aus Programmierkomfort und Rechnerausnutzung, in einer Vielzahl der Systementwicklungen die Sprache PL/M zum Einsatz.

Auch beim Problem der Beschreibungsbank wurde zunächst an eine vollständige Realisierung in PL/M gedacht, zumal diese Sprache recht maschinennah 8-Bit- und 16-Bit-Datentypen und darauf angewandte arithmetische und logische Operationen unterstützt. Die auf dieser Basis recht geschickt formulierbaren Stringoperationen ließen schnelle Ausführungszeiten erwarten. Die Praxis zeigte jedoch, daß bei Bearbeitung großer Datenmengen in Verbindung mit den recht hohen Zugriffszeiten auf die Floppy-Disk-Massenspeicher und der Realisierungssprache PL/M kurze Systemreaktionszeiten nicht immer gesichert waren. So wurden einige kritische Routinen (Umkopieren, Vergleiche...) als Assemblerunterprogramme realisiert.

8085-Assembler und PL/M-Compiler erzeugen standardmäßig relativen Code. Ein zu den Systemdiensten gehörender Binder kann einzeln übersetzte Objektmodule montieren und ihre gegenseitigen Externreferenzen zuordnen. Ein Locator erzeugt danach den zum Ablauf auf dem Prozessor 8085 notwendigen absoluten Code.

Zum Modultest steht ein recht komfortables hardwareunterstütztes Debug-System (/INT 77c/) zur Verfügung, das unter anderem einen Speicherschutz, Haltepunkte und einen Backtrace auf Assemblerebene anbietet.

6.2. Die Arbeit mit dem Beschreibungsbanksystem SCROLL

6.2.1 Die Dialogstufen

Bild 6.2.1 zeigt schematisch den grundsätzlichen Dialogablauf bei der Arbeit mit dem System SCROLL.

Die Beschreibungsbank SCROLL wird zunächst - wie jeder andere Operator - aus der Kommandoebene des Betriebssystems ISIS-II durch Aufruf ihres Namens gestartet. Dieser Startaufruf ist zunächst unabhängig von den zu bearbeitenden Daten. Mit einer weiteren obligaten Angabe beim Start ist also der Name des Softwaresystems festzulegen, dessen Beschreibungsdaten im folgenden Dialog bearbeitet werden sollen.

Vor Dialogbeginn stellt das gestartete System intern fest, ob es als Master-System oder als Slave-System läuft und welche Daten daraus folgend Zugriffsbeschränkungen unterliegen (vgl. die Beschreibung des Befehls INITIALISIERE). Nun erfolgt die erste Meldung auf dem Dialoggerät und die Nachfrage nach Befehlen.

Ein Dialog mit dem System SCROLL ist zerlegbar in eine Folge von Benutzerbefehlen. In der Regel wird jeder Befehl direkt ausgeführt und kehrt in die Ebene der Befehlseingabe zurück. Nach Bild 6.2.1 gibt es jedoch, abhängig vom jeweiligen Befehl, verschiedene typische Ausführungsformen:

Zunächst sind Befehle vorstellbar, deren Ausführung selbständig abläuft. Sie greifen auf vorhandene Beschreibungsdaten zu, modifizieren oder bereiten diese in der Regel auf und geben das entstehende Ergebnis auf dem Drucker aus oder legen es in einer Massenspeicherdatei ab. Auf ihren Ablauf kann von Benutzerseite kein Einfluß genommen werden, allenfalls ist es möglich, ihre Bearbeitung vorzeitig abzubrechen ("Break-Character"), was natürlich zu unvollständigen bruchstückhaften Ergebnissen führt.

Eine zweite Kategorie Befehle hat informierenden Charakter. Auch sie greifen auf vorhandene Beschreibungsdaten zu, erzeugen jedoch seitenweise aufbereitete

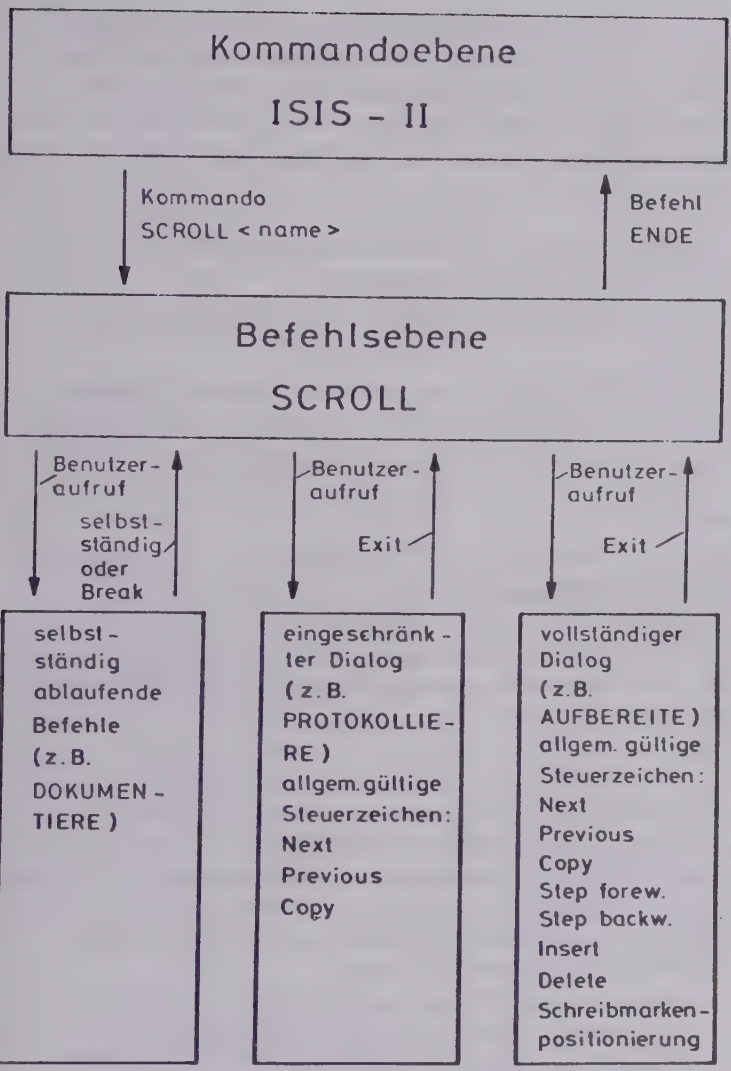


Bild 6.2.1: Die Dialogstufen des Beschreibungsbanksystems SCROLL

Ausgaben dieser Daten auf dem Dialoggerät und erlauben die Steuerung der Ausgabe. Diese Art der Kommunikation mit dem Benutzer soll als "eingeschränkter Dialog" bezeichnet werden; ein Einfluß besteht lediglich auf den Ablauf der Ausgabe. Unabhängig vom speziellen Befehl sind allen eingeschränkten Dialogen vier Steuerzeichen gemeinsam, die eine feste Bedeutung haben:

- der Next-Character — führt zur Vorlage der nächsten Bildschirmseite
- der Previous-Character — führt zur Vorlage der vorherigen Bildschirmseite
- der Copy-Character — veranlaßt eine 1:1-Kopie der aktuellen Bildschirmseite auf dem Schnelldrucker
- der Exit-Character — bricht den Dialog ab und führt zur Rückkehr in die Befehlsebene.

Charakteristikum aller Befehle der dritten Kategorie ist ihre Fähigkeit, Daten in die Beschreibungsbank einzutragen oder eingetragene Daten zu manipulieren. Die Ausführung dieser Befehle erfolgt in einem "vollständigen Dialog" mit dem Benutzer. Unabhängig vom speziellen Befehl werden alle diese Updates in einer Menü-Technik durchgeführt, deren allgemeine Eigenschaften den folgenden Regeln genügen:

- jedes Menü besteht aus einer Folge von Bildschirmseiten (im folgenden "Karten" genannt), die in speziell aufbereiteter Form vom System vorgelegt werden. Wie im eingeschränkten Dialog gibt es den Next-Character und den Previous-Character zum Blättern in den Karten, den Copy-Character zur Hardcopy der aktuellen Karte und den Exit-Character zum Abbruch des Dialogs.
- jede Karte besteht aus einem Kartenkopf, der allgemeine Information über die vorliegenden Datenbereiche enthält und dem eigentlichen Karteninhalt.
- in jedem Karteninhalt gibt es systemeigene Bereiche und benutzereigene Bereiche. Jeder systemeigene Bereich dient zur allgemeinen Benennung und Erläuterung des ihm folgenden benutzereigenen Bereichs, der die eigentliche Information

aufnimmt. Kartenkopf, systemeigene Bereiche und - soweit schon vorhanden - benutzereigene Bereiche werden beim Kartenaufbau vom System eingetragen. Kartenkopf und systemeigene Bereiche sind im folgenden nicht modifizierbar.

Die aktive, initiative Rolle beim nun folgenden Dialog obliegt dem Benutzer. Diese Festlegung wurde mit der Voraussetzung getroffen, daß der Anwender der Beschreibungsbank im allgemeinen kein DV-Laie ist, genau weiß, welche Eintragungen er vornehmen will und aus diesem Grund nicht vom Rechner geleitet werden muß. Der Mensch in diesem Mensch-Maschine-Dialog bestimmt also, welcher Eintrag zu welcher Zeit durchgeführt wird.

Eine Schlüsselrolle im vollständigen Dialog spielt die Bildschirmschreibmarke (Cursor). Sie zeigt dem Benutzer zu jeder Zeit an, an welcher Stelle er einen Eintrag durchführt. Der Cursor ist mit fünf speziellen Zeichen zur Schreibmarkenpositionierung beliebig im gerade aktuellen benutzereigenen Bereich verschiebbar. Neben dem Neueintrag von Information ist es damit möglich, beliebige schon existierende Zeichen zu überschreiben und damit zu korrigieren. Zwei weitere Zeichen mit besonderer Bedeutung unterstützen diese Fähigkeit: Der "Delete-Character" löscht das Zeichen auf der aktuellen Cursor-Position, der "Invert-Character" fügt auf der aktuellen Cursor-Position ein Leerzeichen ein.

Das Dialogsystem der Beschreibungsbank verhindert ein Verlassen des aktuellen benutzereigenen Bereichs mit der Schreibmarke. Vielmehr bleibt das Verlassen dieses Bereichs zwei weiteren speziellen Zeichen vorbehalten. Der "Step-forward-Character" und der "Step-backward-Character" erlauben eine Anforderung an das Dialogsystem, die Schreibmarke an den Beginn des nächsten bzw. des vorherigen benutzereigenen Bereichs zu setzen. Bevor dieser Anforderung stattgegeben wird, wird jedoch der Inhalt des aktuellen Bereichs auf seine syntaktische Richtigkeit überprüft. Hier ergeben sich drei Möglichkeiten:

- der aktuelle Bereich ist leer. Es findet keine Überprüfung statt, das System macht sich intern einen Leervermerk und führt die Anforderung aus.
- der Inhalt des aktuellen Bereichs ist syntaktisch richtig gemäß der für diesen Bereich geltenden Regeln. Der Inhalt wird akzeptiert und die Anforderung ausgeführt.

- der Inhalt des aktuellen Bereichs wird als syntaktisch falsch erkannt. Die Anforderung wird ignoriert und unter der Annahme, daß der Fehler mit hoher Wahrscheinlichkeit sofort durch den Benutzer erkannt wird, erfolgt zunächst nur eine akustische Fehlermeldung. Nun kann ein erkannter Fehler mit den schon beschriebenen Mitteln sofort beseitigt werden. Ist der Fehler jedoch nicht sofort ersichtlich, so dient als Hilfe ein weiteres, letztes allgemeingültiges Sonderzeichen. Der "Question-Character" veranlaßt explizit eine Syntaxüberprüfung über den Inhalt des aktuellen Benutzerbereichs. Das Ergebnis dieser Prüfung erscheint als kurze Meldung in einer Zeile des Kartenkopfes; bei mehreren Syntaxfehlern wird nur der zuerst entdeckte Fehler protokolliert.

  Bleibt der Fehler trotz dieser Hilfe für den Benutzer weiterhin unersichtlich, so kann er das Dialogsystem durch eine zweite Anforderung mit dem Step-forward- oder Step-backward-Character zwingen, die fehlerhafte Eingabe zu akzeptieren und der Anforderung stattzugeben. Diese nur zur Rettung großer Benutzerdatenbereiche gedachte vorübergehende Abspeicherung fehlerhafter Daten wird jedoch im System explizit vermerkt und bei nächster Gelegenheit (z.B. bei erneuter Vorlage der Karte) protokolliert.

Das beschriebene Dialogsystem reagiert für alle dialogführenden Befehle in der gleichen Art, speziell sind befehlsunabhängig die genannten Sonderzeichen immer gleich und in keiner anderen Bedeutung verwendbar.

Mit dieser Voraussetzung sollen im folgenden die existierenden Befehle klassifiziert und in logische Gruppen eingeteilt werden.

6.2.2 Aufgabenbereiche und Befehlstypen

Die Aufgaben im Zusammenhang mit der Beschreibungsbank SCROLL sind von unterschiedlicher und vielfältiger Art. Bevor einige von ihnen näher vorgestellt werden, soll zunächst eine Klassifizierung durchgeführt werden. Ihr Ziel sei die Zuordnung aller möglichen Benutzerbefehle zu einem von sechs möglichen Befehlstypen. Jeder Befehlstyp umfaßt damit alle Befehle, die logisch einem bestimmten übergeordneten Aufgabenbereich angehören.

Im einzelnen soll zwischen den folgenden Befehlstypen und den zugehörigen Aufgabenbereichen getrennt und unterschieden werden:

- Aufgabenbereich Verwaltungsdienste (1)
- Aufgabenbereich Aktualisierungsdienste (2)
- Aufgabenbereich Protokolldienste (3)
- Aufgabenbereich Auswertungsdienste (4)
- Aufgabenbereich Generierungsdienste (5)
- Aufgabenbereich Plotdienste (6)

Zu jedem Aufgabenbereich gehört der entsprechende Befehlstyp, der mehrere konkrete durch den Benutzer aufrufbare Befehle zur Verfügung stellt. Jedoch ist es durchaus möglich, daß der gleiche Befehlsaufruf je nach Benutzer andere Reaktionen zeigt, speziell gibt es bestimmte Befehle, die nur auf dem Master-System laufen und deren Ausführung für alle anderen Benutzer verweigert wird.

Betrachtet werde zunächst der Aufgabenbereich (1). Er umfaßt Systemdienste verwaltender Art, die der allgemeinen Steuerung der Daten und des Systems dienen.

Die Aktualisierungsdienste (2) umfassen Befehle, die eine Modifikation der Beschreibungsdaten erlauben, sie sind jedoch für den Einzelbenutzer nach den schon beschriebenen Regeln eingeschränkt auf die Systemteile, die ihn selbst direkt oder angrenzend betreffen. Nur der Projektsekretär am Master-System kann hier jederzeit den vollen Befehlsumfang abrufen.

Eine Ausgabe aller existierenden Daten auf Konsolbildschirm oder Drucker erlauben die Protokolldienste. Ihre Benutzung ist in keiner Form eingeschränkt; jederzeit ist jeder Benutzer in der Lage, sich über das gesamte System zu informieren. Protokolliert werden nur Daten, die irgendwann in gleicher Form in die Beschreibungsbank eingegeben wurden, eine Bearbeitung besteht lediglich in deren optisch verwertbarer Form für die Ausgabe.

Im Gegensatz dazu erzeugen die Auswertungsdienste (4) aus den existierenden Daten neue Ergebnisse. Ihre Erscheinungsform kann diejenige einer Tabelle sein, in der die Daten nach komplexen Kriterien neu geordnet erscheinen, es sind jedoch auch baumartig strukturierte Listen oder auch Meldungen über das Ergebnis einer Auswertung möglich. Die entstehenden Ergebnisse finden Anwendung für Plausibilitätskontrollen, Dokumentationsvervollständigung und Wartung.

Auch die Generierungsdienste (5) erzeugen neue Ergebnisse durch die Bearbeitung existierender Daten. Ihr Verwendungszweck liegt jedoch in deren sofortigem Weitereinsatz beim Produktionsfortschritt, speziell in der Codierungsphase. Ziel der Generierungsdienste ist eine Einsparung mechanischer Schreibarbeit durch die halbautomatische Erzeugung von Programmcoderahmen und Kommandofolgen und damit eine Unterstützung und Entlastung der Programmierer.

Als einzige setzen die Plotdienste (6) die Existenz eines automatischen Zeichengeräts mit Rechnerschnittstelle (Plotter) voraus. Die Beschreibungsbank ist ohne ihre Anwendung vollständig einsetzbar, jedoch hat ihr Einsatz den Vorteil der besseren Darstellungsmöglichkeit bestimmter Strukturen, deren Ausgabe in Listenform auf einem druckenden Gerät relativ betrachtet unvorteilhafter erscheint.

Aufbauend auf diese allgemeine Übersicht und Einteilung der Möglichkeiten, die die Beschreibungsbank bietet, ist es nun leicht, die in sich abgeschlossenen Aufgabenbereiche getrennt zu betrachten.

6.2.3 Allgemeine Verwaltungsdienste

Jedes größere dialogorientierte Programmsystem benötigt zu seiner Funktion einige Befehle verwaltungstechnischer Art. Befehle dieser Kategorie für die Beschreibungsbank sollen hier zuerst besprochen werden, weiterhin sind unter diesen Abschnitt einige Sonderbefehle eingeordnet, die nicht allgemein zur Verfügung stehen.

Der einfachste, hier der Vollständigkeit halber zu nennende Befehl - er heißt ENDE - führt zum Abschluß des Dialogs und zur Rückkehr in die Kommandoebene des Betriebssystems. Der Befehl SYSTEM schaltet im Dialog ohne den Umweg des Neustarts die Beschreibungsbank auf die Beschreibungsdaten eines anderen Systems um.

Alle nun noch in diesem Abschnitt genannten Befehle haben Sonderaufgaben und sind nur am Master-System ausführbar.

Der Befehl HOLE ermöglicht das Einkopieren von Beschreibungsdaten aus einem anderen System; verwendbar vorteilhaft dann, wenn ein existierendes Modul vollständig unverändert übernehmbar ist. Das so einkopierte Modul wird

keinem Programmiererbereich zugeordnet. Auf diese Weise ist auch der Aufbau von Pseudosystembeschreibungen möglich, die nur als eine Art Beschreibungsbibliothek dienen.

Die Archivbefehle ARCHIVIERE und ARCHIVPROTOKOLL stellen Funktionen zur Verfügung, mit denen der Projektsekretär effektiv für jedes Modul seine Quelle, sein Objekt und seine Beschreibung ins Archiv kopieren kann. Das System verwaltet in seinem Archivverzeichnis das aktuelle Datum, die Versionsnummer und die Namen von Quelldatei, Objektdatei und zugehörigem Datenträger. Nach erfolgter Archivierung wird die Versionsnummer der im System verbleibenden Beschreibung um 1 erhöht. Ein Protokoll des Gesamtarchivs oder der Lebensgeschichte eines Moduls ist abrufbar. RESTAURIERE stellt die letzte Version aus dem Archiv wieder zur Verfügung.

Die nach den beschriebenen Vorüberlegungen endgültig gewählte Verwaltung der Master/Slave-Struktur soll näher beschrieben werden. Der Neuaufbau eines Beschreibungsbanksystems ist nur vom Master-System mittels des Befehls INITIALISIERE möglich. In dem folgenden vollständigen Dialog trägt der Projektsekretär zunächst die allgemeinen Kenndaten des neuen Systems ein und legt dann die in der Systemplanung erarbeitete Hierarchie fest. In der Regel ist das System zu diesem Zeitpunkt nur grob strukturiert, Programmiererbereiche bestehen oft nur aus einem Modul und Schnittstellen sind unvollständig abgeklärt. Durch die Definition der Programmiererbereiche erfährt das System die Namen aller beteiligten Programmierer; im Anschluß an den Initialisierungsdialog werden soviele auf den jeweiligen Namen eines Programmierers lautende Slave-Systeme auf ebensovielen Disketten erstellt, wie die Anzahl der Programmierer vorschreibt. Auch Änderungen in den Programmiererbereichen oder der Anzahl der Programmierer, die sich während der Produktion ergeben, lassen sich durch diesen Befehl bearbeiten.

Der nach Konzept der Beschreibungsbank in regelmäßigen Abständen notwendige Sammellauf wird am Master-System durch den Befehl MISCHE durchgeführt. Nach jeweiliger Anforderung werden nacheinander alle Slave-Systeme eingelesen, auf Änderungen überprüft, die schon beschriebenen Konsistenzprüfungen durchgeführt und so ein neues aktuelles Master-System erstellt. Nach einem Situationsbericht entstehen direkt im Anschluß daran die neuen Slave-Systeme. Damit ist ein lückenloses Netz der automatischen Dokumentation, deren Prüfung und Sammlung und, damit zwangsläufig verbunden, der Verfolgung des Projektfortschritts aufgebaut.

Zu erwähnen bleiben einige Befehle, die zentrale Definitionen für das gesamte Beschreibungsbanksystem festlegen. Als Beispiel sei der schon bekannte Befehl zur Definition eines neuen Datentyps genannt, der zentral für das gesamte System eine Erweiterung der Standarddatentypen vornimmt.

6.2.4 Aktualisierungs- und Protokolldienste

Die wichtigsten Befehle für die einzelnen Programmierer stellen die informierenden und datenmanipulierenden Dienste dar. Sie werden in unmittelbarer Weise täglich von jedem Programmierer angewendet und sind so die wichtigste Benutzerschnittstelle. Protokollierende und manipulierende Befehle unterscheiden sich nur in der Art ihres Dialogs, es werden die gleichen Menükarten verwendet. Der Befehl PROTOKOLLIERE dient zur umfassenden Information über alle einzeln spezifizierbaren Beschreibungsbankdaten auf dem Bildschirm des Dialoggeräts. DOKUMENTIERE erzeugt diese Information in Seitenstrukturierung auf dem Schnelldrucker.

Der Befehl AUFBEREITE führt bei gleicher Spezifikation zu einem vollständigen Dialog, seine Ausführung wird jedoch von vornherein abgelehnt, wenn der Name des aufgerufenen Programmiererbereichs nicht mit dem Namen des Slave-Systems übereinstimmt.

Bewußt wurde auf eine Trennung in eintragende, löschende und korrigierende Befehle verzichtet, dem Benutzer sind durch die Steuerzeichen des vollständigen Dialogs jederzeit alle genannten Möglichkeiten gegeben. Dieses Vorgehen erhöht nicht nur Komfort und Benutzerakzeptanz, sondern vereinfacht auch den Aufbau des Systems SCROLL von Benutzerseite gesehen. Höhere Anforderungen werden jedoch an die Aufmerksamkeit des Benutzers gestellt, da die jederzeit zur Verfügung stehenden Funktionen auch unbeabsichtigte Datenzerstörung zur Folge haben können.

Zur Komplettierung der Datenmanipulationsmöglichkeiten fehlt nun noch die Bearbeitungsmöglichkeit der hierarchischen Baumstruktur. Diese Lücke wird durch den Befehl ERSETZE geschlossen, mit dem - allerdings nur im eigenen Programmiererbereich - die schon in Bild 5.2.2 gezeigten Grundoperationen auf die existierende Struktur anwendbar sind. Der Befehl führt zu einem vollständigen Dialog, in dem die Verteilung der Schnittstellen geklärt wird. Für seinen eigenen Programmiererbereich darf somit jeder Programmierer den Systemaufbau nach seinem Belieben wählen, es wird nur die Einhaltung der Baumstrukturregeln gewährleistet.

6.2.5 Auswertende Dienste

Auch ohne weitergehende Auswertung stellt sicher schon die reine maschinelle Verwaltung der Beschreibungsbankdaten eine gute Unterstützung der Softwareproduktion und -dokumentation dar. Nach dem ersten Schritt der Speicherung bietet sich jedoch als zweiter Schritt die weitergehende Auswertung der Information an.

Auswertende Dienste werden sowohl für den Projektsekretär wie auch für die einzelnen Programmierer angeboten. Im Interesse des Projektsekretärs liegt die automatische Verfolgung des Produktionsverlaufs. Der zuständige Befehl SITUATION ist wie üblich auf das Master-System beschränkt. Mit der Kenntnis des aktuellen Datums und des in den allgemeinen Modulbeschreibungen verwalteten Bearbeitungszustands eines jeden Moduls (vgl. 5.2.5.1) ist jederzeit ein Situationsbericht organisatorischer Art möglich. Er enthält die aktuellen Zustände aller Module, überprüft die Einhaltung von Sollterminen und weist auf Terminüberschreitungen hin. Durch die Wahl der einzelnen Produktionsmarken, die in etwa gleichen Zeitabständen erreichbar sind, werden Abweichungen vom geplanten Produktionsverlauf frühzeitig aufgedeckt.

Eine allgemein interessierende Auswertung betrifft den Aufbau und die Ausgewogenheit der Hierarchiestruktur und damit die Abhängigkeit der einzelnen Module voneinander. Der Befehl HIERARCHIE ermöglicht die Ausgabe der gesamten Baumstruktur in Listenform mit zugehöriger Durchnumerierung nach der gewählten Nomenklatur; auch der Aufbau einzelner Bäume, der Aufbau von Unterstrukturen mit vorgebbarer Wurzel und die Struktur im logischen Umkreis um ein Modul ist bearbeitbar.

Weiterhin wünschenswert ist eine Aufstellung aller Variablen oder eines ausgewählten Teils in Form einer Cross-Referenz-Liste; zu jedem Variablennamen werden alle Module aufgeführt, die Zugriff auf deren Inhalt haben, der schreibende Zugriff wird speziell gekennzeichnet. Der zugehörige Befehl CROSSREF ermöglicht zusätzlich bei Vorgabe von Modulnamen die Ermittlung aller Variablen, die mehr als einem Modul zur Verfügung stehen. Die Angabe von Modul- und Variablennamen bildet die mengentheoretische Schnittmenge; die Überprüfung der modulgemeinsamen Variablen wird auf die explizit genannten Namen beschränkt. Speziell bei der Fehlersuche findet die Cross-Referenz-Liste ihren Einsatz: Mit ihrer Hilfe ist schnell feststellbar, welche Module Zugriffsberechtigung zu einer fehlerhaften Datenstruktur haben.

Eine ähnliche Liste ist für die dateibearbeitungsbezogenen Operationen erstellbar; aus ihr geht mit einem Blick hervor, welche Dateien in welcher Form von welchen Modulen bearbeitet werden.

Eine letzte Auswertung ist in Planung. Mit Hilfe der gespeicherten Information über die Aufrufmöglichkeiten in einem beschriebenen echtzeitorientierten System soll eine Analyse über dessen Real-Zeit-Verhalten durchgefunrt werden. Die erwarteten Ergebnisse sollen Gefahren aufzeigen, die sich aus der gewählten Prozeßstruktur ergeben, so sind etwa mögliche Systemverklemmungen oder außergewöhnliche Systembelastungen erkennbar und analysierbar.

Die auf Grundlage der Dokumentationsdaten durchführbaren Auswertungen stellen somit für die Programmierer und den Projektleiter eine weitere Hilfe zur Produktion, Wartung und Systempflege dar.

6.2.6 Generierende Dienste und Plotdienste

Die im folgenden beschriebenen gehobenen Dienste der Beschreibungsbank befinden sich zum aktuellen Zeitpunkt im Planungstadium.

Unter Berücksichtigung der Tatsache, daß durch die Erstellung von Dokumentation und Quellprogramm Information entsteht, die durch ihr Auftreten in beiden Schriftstücken Redundanz enthält, besteht die prinzipielle Möglichkeit, Teile des Quellprogramms aus der Dokumentation automatisch zu gewinnen. Ein Vorteil für die Programmierer liegt dann in deren Arbeitserleichterung durch Verringerung oder Vermeidung doppelt durchzuführender Schreibarbeit.

Ein erster und einfach realisierbarer Schritt besteht in der automatischen Generierung eines Standard-Programmkopfes, der als Kommentar der jeweiligen Programmiersprache allgemeine Modulinformation enthält. Die hier abgelegten Daten stellen einen Auszug aus den in der Beschreibungsbank enthaltenen allgemeinen Modulkenndaten dar und werden in den Quellprogrammkopf dupliziert. Mit ihrer Hilfe ist jedes Quellprogramm und jedes Listing eindeutig identifizierbar; eigentlich eine Selbstverständlichkeit, aber geschätzt von jedem Praktiker, der die Probleme fremder unbezeichneter Quellen kennt.

Mit der in der Beschreibungsbank abgespeicherten Modulschnittstellen-Information und der Kenntnis der Realisierungsprogrammiersprache ist es weiterhin

möglich, eine automatische Generierung aller Inter-Modul-Deklarationen durchzuführen. Diese Programmgenerierung ist stark programmiersprachenabhängig. In einer ersten Stufe der Entwicklung ist deshalb die konkrete Unterstützung von FORTRAN, PL/M und PASCAL geplant, was zum einen einem Querschnitt durch die Generationen der Programmiersprache entspricht und zum anderen die Standardprogrammiersprachen am Implementierungsrechner unterstützt.

Der Generierungsdienst der Modulschnittstellen umfaßt sowohl die gemeinsamen Datenstrukturen wie auch die Schnittstellen im Programmablauf. Diese Unterstützung erfordert für die verschiedenen Programmiersprachen verschieden hohen Aufwand.

In FORTRAN werden Datenschnittstellen in Form von COMMON-Blöcken angelegt. Damit muß der Name jedes COMMON-Blocks, sein innerer Aufbau und die Typendeklaration jeder COMMON-Date aus dem Satz der in FORTRAN zugelassenen Standarddatentypen erzeugt werden. Für die Programmschnittstellen genügt die Erzeugung der EXTERNAL-Deklarationen für Unterprogramme vom Typ FUNCTION.

In PL/M besteht die vollständige Datenschnittstellendeklaration aus der Aufzählung aller Variablennamen mit Nennung des Datentyps aus der Menge der PL/M-Standarddatentypen und des Attributs EXTERNAL oder PUBLIC. Alle in der tieferen Modulhierarchie verwandten Unterprogramme erfordern eine Externdeklaration unter Angabe ihrer vollständigen Parameterliste und deren Typen.

Im Gegensatz zu PASCAL, das eigene Datentypdeklarationen zuläßt, können bei FORTRAN und PL/M Probleme bei der Generierung auftreten, wenn der in der Dokumentation beschriebene Datentyp oder die Datenstruktur in dieser Form in der gewählten Programmiersprache nicht zulässig ist. In diesem Fall muß bei der Generierung eine Abbildung dieser Daten auf eine erlaubte Darstellung erfolgen. Der Programmierer wird unterstützt durch einen Bericht, in dem diese Zuordnung beschrieben wird.

Analog zu den Datendeklarationen ist es schließlich auch möglich, einen Programmcoderahmen für jedes Modul zu erzeugen. Voraussetzung hierfür ist die Verwendung der in 5.2.5.3 beschriebenen Schlüsselworte, die den Programmablauf charakterisieren. Ist diese Struktur als fehlerfrei erkannt worden, so kann - wieder mit der Kenntnis der gewählten Programmiersprache - maschinell ein Coderahmen erstellt werden, der alle logischen Ablaufstruk-

turen in ihrer endgültigen Form enthält. Alle übrigen Programmteile, etwa Bedingungen oder Blockinhalte, bleiben zunächst - wie in der Beschreibungsbank - in Form von verbalem Text stehen. Mit Hilfe eines bildschirmorientierten Texteditors kann nun der Programmierer Zeile für Zeile diesen Text durch explizite Statements ersetzen, also den vorgegebenen Programmrahmen ausfüllen.

Die bisher vorgestellten Funktionen der Beschreibungsbank stellen in ihrer Gesamtheit ein mächtiges und in sich abgeschlossenes Werkzeug zur Verfolgung und Dokumentation von Softwareprojekten dar. Speziell die Dokumentation verzichtet jedoch bisher vollständig auf den Einsatz graphischer Mittel. Der Grund dieser Restriktion liegt in der auferlegten Beschränkung der Implementierungsrechnerkonfiguration, die den Einsatz eines maschinellen Zeichengeräts nicht vorsah, und der Feststellung, daß schlechte graphische Darstellungen auf druckenden Geräten den hohen Aufwand ihrer Erstellung nicht rechtfertigen.

Es ist mit den beschriebenen Funktionen gelungen, eine vollständige Systemdokumentation ohne graphische Darstellungen durchzuführen; es bleibt jedoch unumstritten, daß ebendiese graphischen Darstellungen, wo immer sie möglich sind, wegen ihrer Anschaulichkeit den Dokumentationswert erhöhen. Aus dieser Überlegung resultieren die Plotdienste, deren Verwendung als Option die Konfigurationserweiterung um einen Plotter voraussetzt.

Eine erste graphisch darstellbare Struktur umfaßt die Darstellung der Systemhierarchie. Dabei bietet sich sowohl die zeichnerische Darstellung des Gesamtbaums wie auch - als Ausschnitt - beliebiger Unterbäume oder Teilbäume an. Auch die Grenzen der Programmiererbereiche sind vorteilhaft markierbar. Die Einzelbildgröße ist auf DIN A 4 (quer) voreingestellt und kann auf die für Druckerendlospapier typische 12"-Größe umgeschaltet werden. Vorgesehen ist eine automatische Aufspaltung in Unterstrukturen bei Seitenüberlauf mit dem Kriterium, möglichst vollständige Unterbäume auf einer Seite unterzubringen. Die Beschriftung beschränkt sich für jedes Modul in seinem Namen und dem Namen seines Programmierers.

Der Programmablauf in den einzelnen Modulen bietet sich besonders vorteilhaft zur graphischen Darstellung an. Das Kriterium der Syntaxfehlerfreiheit gemäß der in 5.2.5.3 aufgestellten Grammatik beinhaltet die Möglichkeit der konfliktfreien graphischen Darstellung in der von Nassi und Shneiderman vorgeschlagenen Nomenklatur. Jede syntaxfehlerfreie teilformale Programmablaufbeschreibung eines Moduls ist also prinzipiell als Nassi/Shneidermann-Dia-

gramm darstellbar. Dabei erfahren die formalen Schlüsselworte der Beschreibung eine Transformation in die graphischen Symbole dieser Technik, der verbindende verbale Text wird unverändert übernommen und stellt den Symbolinhalt dar. In der Praxis ist zu berücksichtigen, daß eine in der Beschreibungssprache ohne Schwierigkeiten mögliche Verschachtelung in ihrer graphischen Darstellung wegen des endlichen Papierformats nicht möglich sein kann. Unbedingt notwendig ist also die automatische, zur Not mehrstufige, sinnvolle Aufspaltung in Teildiagramme mit entsprechenden Verweisen.

Geplant ist es schließlich, für die Dokumentation von Real-Zeit-Systemen die parallelen Vorgänge in Form von Petri-Netzen graphisch sichtbar zu machen. Diese weniger bekannte Darstellungsart soll dazu beitragen, die oft komplexen dynamischen Abläufe des beschriebenen Systems übersichtlicher und durchschaubarer zu machen.

Die angebotenen Plotdienste runden eine Systemdokumentation sicher ab, die von ihr erzeugte Information ist jedoch streng betrachtet redundant. Es sollte daher dem jeweiligen Beschreibungsbankanwender überlassen bleiben, die Anschaffung eines Plotters gegen die anschaulich damit erstellbare Zusatzdokumentation abzuwiegen.

6.3 Der innere Aufbau der Beschreibungsbank

6.3.1 Allgemeine Gesichtspunkte zur Struktur

Im bisherigen Teil dieser Arbeit ist die Beschreibungsbank aus Sicht des Benutzers betrachtet worden. Vor einer konkreten Realisierung sind jedoch einige Gedanken aus der Betrachtungsrichtung des Erbauers sinnvoll, denn eine allgemeine Analyse der Aufgabe und der Umgebungsbedingungen aus Sicht des Datenverarbeiters hat entscheidenden Einfluß auf die innere Struktur des Systems.

Die Beschreibungsbank dient einer Zusammenfassung spezieller Datenbestände als Basis eines Informationssystems. Damit erfüllt sie die wesentlichen Kriterien einer Datenbank, die nach Ahrens /AHR 71/ "ein formalisiertes Informationssystem in einem abgeschlossenen Informationsbereich darstellt und die wesentliche Grundlage für die Informationswiedergewinnung bildet". Datenbankmodelle unterscheiden sich grundlegend durch die Wahl der Organisationsform der Datenbestände im Zentralspeicher und auf den Externspeichern und

durch die daraus resultierenden Zugriffspfade. Je nach Art der Aufgabe und der Umgebungsbedingungen sind verschiedene Abhängigkeiten verschieden stark zu gewichten, die der einen oder anderen Organisationsform den Vorzug geben.

Es ist im Rahmen dieser Arbeit sicher nicht möglich, auch nur annähernd vollständig auf die verschiedenen Datenbankmodelle und ihre Vorteile einzugehen. Vielmehr sollen im folgenden einige Anforderungen genannt und - speziell auch vor dem Hintergrund der gewählten Hardwarekonfiguration - bewertet werden, die die Kriterien zur Wahl einer Struktur liefern sollen. Aus der geforderten Zugriffszeit zu den verwalteten Daten resultieren die ersten Bedingungen an deren Struktur. Nach den schon in Abschnitt 5.1.2 durchgeführten Überlegungen von Benutzerseite ist hier jedoch zu differenzieren. So sollte besonderer Wert auf schnelle Ausführungszeiten für Protokolldienste gelegt werden; für Änderungsdienste dagegen ist diese Forderung nicht so streng zu formulieren. Speziell hohe Anforderungen sind an die Zugriffszeit zu den verwalteten Modulnamen, Prozedurnamen und Variablennamen zu stellen, denn diese bilden als Schlüssel die Ausgangspunkte aller Zugriffspfade zu den zugehörigen Beschreibungen und erfahren bei Eintrage- und Änderungsdiensten eine häufige Durchsuchung zur Vermeidung von Doppeleinträgen.

Aufgrund des relativ kleinen Implementierungssystems sind gezielte Überlegungen zur Speicherplatznutzung erforderlich. Es ist aus Platzgründen soweit wie möglich auf Redundanzfreiheit der abgespeicherten Information zu achten. Jeder mehrfach verwendete Beschreibungsteil sollte nur einmal vollständig erfaßt und durch geeignete Verweise zugänglich gemacht werden. Als Beispiel sei die Beschreibung einer mehrerer Module gemeinsamen Variablen genannt, die nur einmal zentral geführt wird und die über ihren Namen erreicht werden kann.

Spezielle Probleme an die Speicherplatznutzung stellen die Änderungsdienste. Es ist zu vermeiden, daß die Originale korrigierter oder erweiterter Beschreibungsteile als "Leichen" in den Dateien liegenbleiben; der freigewordene Platz sollte vielmehr sobald wie möglich neu verwendet werden. Eine effektive Dateispeicherverwaltung ist hierzu Grundbedingung.

Die Datensicherheit als weiteres Kriterium wird durch die geforderte Redundanzfreiheit eher negativ beeinflußt. So wurde eine mögliche Restaurierung von Daten nach Informationsverlust wissentlich auf eine andere Ebene verlagert. Unter Ausnutzung des Vorteils der durch die Arbeitsweise bedingten

gleichzeitigen Existenz mehrerer Systeme und eines Master-Systems ist der maximale Schaden eines durch widrige Umstände bedingten Systemzusammenbruchs begrenzt auf den Zeitpunkt des letzten Sammellaufs.

Unter Berücksichtigung der gefundenen und sich teilweise widersprechenden Anforderungen an Zugriffszeit, Redundanzfreiheit und Datensicherheit wurde eine Struktur erarbeitet und realisiert, die in ihren Grundlagen im folgenden beschrieben werden soll.

6.3.2 Datenhierarchie und Zugriffswege

Der Aufsatzpunkt für die Organisation einer möglichst wahlfrei zugreifbaren Datenstruktur auf dem Massenspeicher des Implementierungsrechners liegt auf recht niedrigem Niveau. Das unterstützende Betriebssystem bietet die Verwaltung sequentieller Dateien, die für den beabsichtigten Zweck zunächst denkbar ungeeignet erscheinen. Der geforderte wahlfreie Zugriff wird prinzipiell erst möglich durch einen darüberhinaus benutzbaren imaginären Zeiger, der per Systemaufruf beliebig in der Datei verschiebbar ist und über den festgelegt wird, an welcher Stelle die nächste lese- oder Schreiboperation beginnen soll. Im Gegensatz zur normalen schreibenden Bearbeitung sequentieller Dateien bleibt bei dieser Arbeitsweise die Information bis zum Dateiende erhalten; nur die neu geschriebenen Zeichen werden ersetzt.

Ohne großen Aufwand läßt sich auf dieser Grundlage eine indexsequentielle Verwaltung realisieren, mit der Daten fester Satzlänge bearbeitbar sind. Besonders einfach wird diese Struktur, wenn Eintragungen grundsätzlich am Dateiende vorgenommen werden, Korrekturen den alten Eintrag überschreiben und Löschungen durch Löschung des Index vorgenommen werden.

Auch die Verwaltung von Datensätzen beliebiger Länge ist auf Grundlage dieser indexsequentiellen Verwaltung prinzipiell lösbar, indem jeder Datensatz in Untersätze gleicher Länge aufgelöst wird. Die Forderung nach physikalisch hintereinander abgelegter Information ist jedoch so nicht erfüllbar, denn eine Korrektur, die einen bestimmten Datensatz verlängert, führte dann zu umfangreichen Schiebe- und Verwaltungsmaßnahmen. Die beliebige Ablage der Untersätze und deren Verkettung durch Indexzeiger umginge dieses Problem, weist jedoch neben den zusätzlichen Verwaltungsdaten den Nachteil der stark verstreuten Information mit entsprechend vielen Zugriffen des relativ langsamen Massenspeichers auf.

Aus diesen Gründen erweist sich für die Speicherung von Datensätzen ungleicher Länge eine andere Verwaltung als wirkungsvoller. Grundlage hierzu bildet eine Speicherverwaltung, die den Speicherraum einer Datei in ganzzahligen Vielfachen einer Grundgröße vergibt. Die Löschung eines Datensatzes führt bei dieser Methode zum Eintrag des freigewordenen Platzes in eine Freispeicherliste; die Anforderung von Speicherraum zum Neueintrag stellt den Bereich aus der Freispeicherliste zur Verfügung, der nach dem Kriterium der Länge optimal geeignet ist. Nur bei negativem Suchergebnis in der Freispeicherliste wird am Dateiende eingetragen.

Ausgangspunkt aller Datenbearbeitungsdienste in der Beschreibungsbank sind die Namen der beschriebenen Module, Prozeduren und Variablen. Sie stellen den Beginn aller Zugriffspfade zu weiteren auf sie bezogene Daten dar und sorgen durch ihr einmaliges Vorkommen für die Vermeidung von Doppeldeutigkeiten.

Alle existierenden Namen werden aus Gründen der Zugriffszeit im Zentralspeicher gehalten. Als einzige weitere Information wird ein Index mitgeführt, der auf den Ort näherer Angaben auf dem Massenspeicher verweist. Ein Duplikat dieser Namensliste existiert auf dem Massenspeicher und wird nach jedem informationsverändernden Befehl aus Sicherheitsgründen durch vollständige Kopie aktualisiert. Die Namensliste wird stets geordnet geführt; dem Nachteil von Schiebeoperationen beim Neueintrag und bei der Korrektur steht die Anwendungsmöglichkeit effektiver Suchverfahren gegenüber. Das wünschenswerte binäre Suchverfahren bedingt allerdings als weitere Voraussetzung eine feste Länge der Listenelemente. Bei einer maximal erlaubten Namenslänge von 32 Zeichen, die in der Regel nur zu 30....40% genutzt wird, wäre hier der Speicherverschnitt unzulässig hoch. Die realisierte Alternative basiert auf einer dicht gepackten Namensliste und einer zusätzlichen Tabelle mit 25 Zeigern, die jeweils das erste Listenelement angeben, in dem ein Name mit einem neuen Anfangsbuchstaben abgelegt ist. Innerhalb der so bekannten Unterliste wird ein einfaches sequentielles Suchen durchgeführt. Unter der Annahme der statistischen Gleichverteilung aller zulässigen Zeichen in allen Namen wird damit im Mittel ein fünfzigstel der Liste durchsucht, ein Wert, der unter Berücksichtigung der schnellen Zentralspeicherzugriffszeit akzeptabel erscheint.

Bild 6.3.1 zeigt die weitere prinzipielle Struktur der verwalteten Daten.

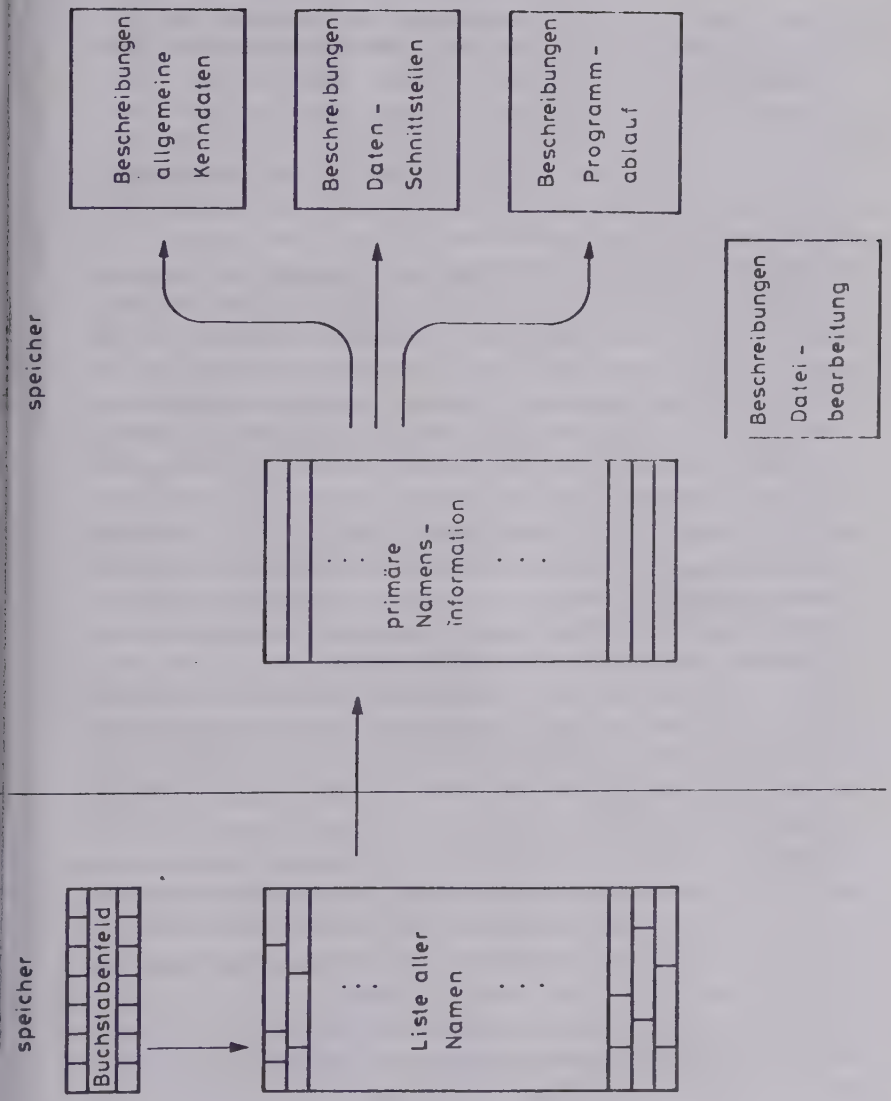


Bild 6.3.1 Realisierte Struktur und Zugriffspfade der in der Beschreibungsbank verwalteten Daten

Nachdem die Namensliste im Zentralspeicher nur die Information über die Existenz eines Namens enthält, sind nähere Einzelheiten über einen Verweis in einer Massenspeicherdatei abrufbar. Die hier abgelegte primäre Namensinformation hat eine fest Satzlänge von 32 byte und ist nach der oben beschriebenen indexsequentiellen Verwaltung organisiert. Ein Neuaufbau und damit eine Bereinigung dieser Datei erfolgt nur bei einem Sammellauf, da wegen der Größe der einzelnen Einträge durch diese Datei in der Zwischenzeit kein Massenspeicherengpaß zu befürchten ist.

In dieser ersten Hierarchiestufe ist Information abgelegt, die den betreffenden Namen näher beschreibt. In Abhängigkeit vom zunächst genannten Typ werden Angaben wie die Hierarchienummer, die Syntaxfehlerfreiheit des Eintrags, die aktuelle Versionsnummer oder ein Korrekturvermerk seit dem letzten Sammellauf gehalten. Ziel dieser primären, durch einstufigen Verweis erreichbaren Beschreibungsdaten ist die vollständige Befriedigung aller Zugriffe durch Befehle, die zu ihrer Ausführung nicht die vollständigen Daten zu einem Schlüssel benötigen.

Ein weiterer Verweis führt von den primären Beschreibungsdaten zur vollständigen existierenden Beschreibung eines Namens. Diese nach ihrer Größe umfangreichste Datenmenge erfährt eine Aufteilung auf drei Dateien; die Zuordnung der Daten geschieht nach der logischen Aufteilung in allgemeine Beschreibungen, Datenbeschreibungen und Ablaufbeschreibungen. Die vom Umfang der existierenden Beschreibungen stark unterschiedlichen Informationsmengen werden nach der oben beschriebenen Dateispeicherverwaltung organisiert. Hierdurch ist bei vertretbarem Aufwand eine optimale Massenspeichernutzung gewährleistet; eine vollständige Bereinigung auch dieser Dateien erfolgt automatisch beim regelmäßigen Sammellauf.

Eine Sonderstellung nimmt die Beschreibung der Dateibearbeitung ein. Die hier verwaltete Informationsmenge ist in der Regel so gering und strukturell so einfach, daß eine Ablage in zwei Hierarchieebenen unnötig erscheint.

Überschlägige Berechnungen und erste Erfahrungen haben gezeigt, daß mit der gewählten Datenstruktur je nach Umfang der Dokumentation die Füllgrenze eines Floppy-Disk-Datenträgers bei 150...200 Modulbeschreibungen liegen dürfte; dies entspricht einem mittleren Softwaresystem mit einem Umfang von 40...50K Objekt.

Die Vergangenheit und die Gegenwart sind unsere Mittel.
Die Zukunft allein ist unser Zweck.

Blaise Pascal

7. Ausblick

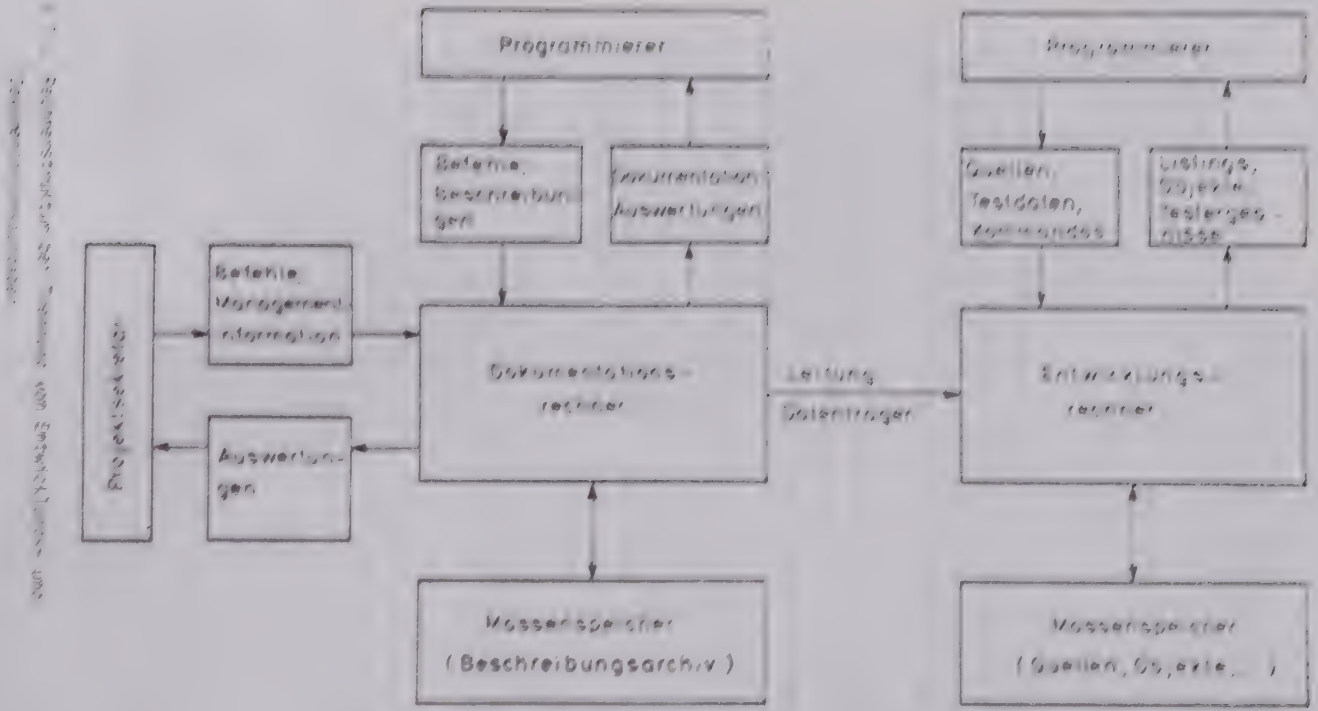
Die vorliegende Arbeit beschreibt ein Softwaresystem - die Beschreibungsbank -, das geeignet ist, die Softwareprojektverfolgung und -dokumentation zu unterstützten. Die Beschreibungsbank läuft als ein mögliches Programm auf einem Mikrorechnersystem, auf dem auch Programmentwicklung durchgeführt wird.

Die zugrundeliegende Arbeitsweise kann, speziell vor dem Hintergrund des Ein-Benutzer-Systems, bei hoher Projektbelastung durch die zusätzlichen Beschreibungsbankeinsätze durchaus zu Engpaßsituationen bei der Rechnerbelegung führen. Eine logische Folge ist die Investition weiterer Entwicklungssysteme, die bei Bedarf von den Programmierern zur Programmentwicklung oder zur Dokumentation eingesetzt werden.

Mit Blick auf die fallenden Hardwarepreise erscheint es in einer solchen Situation als sinnvoll, einen Rechner ausschließlich zu Dokumentationszwecken zu degradieren und damit die Entwicklungsrechner zu entlasten. Bild 7.1 zeigt eine mögliche Struktur.

Unter geringer Modifikation der beschriebenen Arbeitsweise findet bei dieser Architektur die Dokumentation und Projektverfolgung ausschließlich auf dem dafür reservierten Rechner statt. Ein expliziter Sammellauf ist nicht mehr notwendig; vielmehr werden die Daten im Moment ihrer Modifikation sowohl auf dem eigenen Slave-System als auch auf dem im Hintergrund liegenden und nicht direkt zugreifbaren Master-System aktualisiert. Schnittstelleninkompatibilitäten werden im Moment ihres Auftretens erkannt und verhindert. Direkten Zugriff zum Master-System hat weiterhin ausschließlich der Projektsekretär.

Auch in dieser Struktur sind durch die Existenz der Slave-Systeme jederzeit und auf beliebigen Rechnern Protokolldienste durchführbar. Information über beliebige Systemteile ist also weiterhin kurzfristig zu erreichen und bedingt keine Abhängigkeit vom Spezialrechner. Nur die Modifikation der Beschreibungen wird auf den Entwicklungsrechnern verhindert,


Programmierer
Befehle, Beschreibungen
Dokumentation, Auswertungen
Befehle, Vorbesetzungs-information
Programmierer
Auswertungen
Dokumentations-rechner
Leitung, Datenträger
Entwicklungs-rechner
Massenspeicher (Beschreibungsarchiv)
Massenspeicher

Die durch Generierungsdienste im Dokumentationsrechner entstehende und in der Programmentwicklung weiterverwendbare Information bedingt eine Verbindung zwischen Dokumentationsrechner und Entwicklungsrechner. Im einfachsten Fall genügt ein Austausch von Floppy-Disk-Datenträgern, eine Leitungsverbindung ist sicherlich komfortabler und ohne Übertragungsfehler darüber hinaus erweiterbar auf große räumliche Entfernungen.

Die Ausnutzung der angebotenen Datenfernübertragungsmöglichkeiten eröffnet schließlich eine weitere Möglichkeit. Mit der Voraussetzung eines physikalisch kompakten, etwa in einem Aktenkoffer transportablen Rechnersystems ist für den Wartungsingenieur vor Ort durch Übertragen die gesamte aktuelle Dokumentation erreichbar. Diese Art des Zugriffs bei Bedarf sichert die Aktualität der Beschreibungen und vermeidet den Transport von unnötig großen Unterlagen. Erst bei der Fehleranalyse erfolgt die Entscheidung über die benötigte Information; vor Ort sind alle Mittel zur Korrektur eines Fehlers und zur Aktualisierung der Dokumentation vorhanden.

Der letzte Abschnitt gibt einen Eindruck der Möglichkeiten, die durch den Einsatz rechnerunterstützter Projektverfolgungs- und Dokumentationssysteme gegeben sind. Die gute Dokumentation wird in den nächsten Jahren mehr denn je Grundlage qualitativ hochwertiger Software sein. Das vorgestellte Konzept der Beschreibungsbank versteht sich als ein Vorschlag auf dem Weg zur Lösung dieser Probleme.

Literatur

/AHR 71/ Ahrens, W.
Datenbanksysteme
de Gruyter 1971

/ALB 79/ Albert, J., Mokler, A. und von der Wense, E.
Beschreibungsregeln für Software
Siemens AG 1979

/BAK 73/ Baker, F.T. u. Mills, H.D.
Chief Programmer Teams
Datamation 19/1973

/BAU 81/ Baumann, R. (Hrsg.)
Fachtagung Prozeßrechner 1981
Abschnitte Entwurfshilfsmittel I-III
Springer 1981

/BER 80/ Bersoff, E., Henderson, V. u. Siegel, S.
Software Configuration Management
Prentice Hall 1980

/BIE 79/ Biewald, J. u. Goehner, P. u. Lauber, R. u. Schelling, H.
EPOS - A Specification and Design Technique for Computer Controlled Real Time Automation Systems
4. Int. Conference on Software Engineering
München 1979; Konferenzband

/BLA 81/ Blazert, H.
Das Softwareentwicklungssystem PLASMA
Angewandte Informatik 5/1981

/BOE 75/ Boehm, B.W.
The high cost of Software
Addison Wesley 1975

/BRO 75/ Brooks, F.P. jr.
The mythical man-month
Addison-Wesley 1975

/CAL 76/ Callender, E.D.
Industrial practices to control
computer program costs
Aerospace corp., Cal. 1976

/COR 80/ Correll, C.H. u. Debest, X.
Untersuchungen zur Verbesserung
der Software-Produktion
Teil 1-5 (GMD-Bericht 132)
Oldenbourg 1980

/DAH 72/ Dahl, O.-J. u. Dijkstra, E.W.
u. Hoare, C.A.R.
Structured Programming
Academic Press 1972

/DAN 80/ MAPSOFT - A Management Approach to
the Production of Software
Firmenschrift der Fa. Danet GmbH, Griesheim

/DEN 80/ Denert, E. u. Hesse, W.
Projektmodell und Projektbebliothek:
Grundlagen zuverlässiger Software-Entwicklung
und Dokumentation
Informatik-Spektrum 3/1980

/DIN 75/ DIN-Taschenbuch 25:
Informationsverarbeitung
Beuth-Verlag 1975

/ELB 73/ Elben, W.
Entscheidungstabellentechnik
de Gruyter 1973

/FLO 81/ Floyd, C. u. Kopetz, H.
Software Engineering - Entwurf
und Spezifikation
Teubner 1981

/GEI 80/ DARTS - design aid for real
time systems
Firmenschrift der Fa. GEI, Aachen

/GEO 77/ George, U.
Programmierung von Mikrorechnern
Studienarbeit S 26 am Lehrstuhl
für Datenverarbeitung der Ruhr-Universität
Bochum

/GEW 79/ Gewald, K. u. Haake, G. u. Pfadler, W.
Software Engineering
Oldenbourg 1979

/GOO 81/ Goos, G. (Hrsg.)
Werkzeuge der Programmiertechnik
(GI-Arbeitstagung 1981)
Springer 1981

/HAA 81/ Haag, W.
Dokumentation von Anwendungssystemen
aus der Sicht der Benutzer
Toeche-Mittler 1981

/HEI 79/ Heilmann, H. (Hrsg.)
8. Jahrbuch der EDV
Computerunterstützte Systemplanung
und Systemdokumentation
Forkel 1979

/HOM 80/ Hommel, G.
Vergleich verschiedener Spezifikationsverfahren am Beispiel einer Paketverteilanlage, Teil 1 u. 2
Kernforschungszentrum Karlsruhe,
KFK-PDV-Bericht 186

/HUEN 80/ Hünke, H. (Hrsg.)
Software Engineering Environments
North Holland 1980

/HUG 80/ Hughes, J.K. u. Michtan, J.I.
Strukturierte Software-Herstellung.
Anleitung für Programmierer und Manager.
Oldenbourg 1980

/IBM 75/ HIPO - A Design Aid and Documentation Technique
GC 20 - 1851 - 11
White Plains 1975

/INF 76/ Infratest Wirtschaftsforschung GmbH
Stand und Entwicklung des Marktes für mehrfach verwendbare Anwendersoftware in der gewerblichen Wirtschaft
1976

/INT 77a/ 8080/8085 Assembler Language
Programming Manual
Intel Corporation, Santa Clara, Californien
Manual Nr. 9800301D

/INT 77b/ PL/M-80 Programming Manual
Intel Corporation, Santa Clara, Californien
Manual Nr. 9800286

/INT 77c/ ICE-85 In-Circuit emulator operating instructions for ISIS-II-users
Intel Corporation, Santa Clara, Californien
Manual Nr. 9800463B

/INT 78a/ FORTRAN-80 Programming Manual
Intel Corporation, Santa Clara, Californien
Manual Nr. 9800481A

/INT 78b/ BASIC-80 Reference Manual
Intel Corporation, Santa Clara, Californien
Manual Nr. 9800758A

/INT 79/ PASCAL-80 User's Guide
Intel Corporation, Santa Clara, Californien
Manual Nr. 9801015-01

/JAC 75/ Jackson, M.A.
Principles of Program Design
Academic Press 1975

/KAT 76/ Katzan, H.
Methodischer Systementwurf
Verl. Ges. R. Müller 1976

/KOC 78/ Koch, G.R. u. Hoffmann, R.H.
Statistische Angaben zum Stand der Prozeßdatenverarbeitung in der BRD
Angewandte Informatik 6/1978

/KOC 79/ Koch, G.R.
Systematisches Softwareengineering für Mikrocomputer
Elektronik 21 u. 22 1979

/KUP 81/ Kupper, H.
Zur Kunst der Projektsteuerung
Oldenbourg 1981

/LIN 79/ Linger, R.C., Mills, H.D. u. Witt, B.I.
Structured Programming.
Theory and Practice.
Addison-Wesley 1979

/MOL 81/ Molzberger, P. u. Schelle, H.
Software. Moderne Methoden zur Planung,
Realisierung und Kontrollle der Entwicklung.
Oldenbourg 1981

/NAS 73/ Nassi, I. u. Shneiderman, B.
Flowchart Techniques for Structured
Programming
SIGPLAN Notices 8/1973

/NAU 60/ Naur, P.
Documentation Problems:
ALGOL 60
Comm. A.C.M. 5/1960

/ODG 72/ Odgin, J.L.
Designing Reliable Software
Datamation 7/1972

/PAR 72/ Parnas, D.L.
On the Critica to be used in
decomposing systems into modules
Comm. A.C.M. 1972/12

/PAR 74/ Parnas, D.L.
Hierarchical Structure
IFIP Congress 1974 in Amsterdam
North Holland Publ. 1974

/PAR 80/ Parkin, A.
Systems Management
Edward Arnold 1980

/PER 77/ Perrott, R. (Hrsg.)
Software Engineering
Academic Press 1976

/POS 80/ Posa, J.G. und LeBoss, B.
Intel takes aim at the '80s
Electronics 28.2.80, S. 89-95

/PRO 75/ Proceedings of the 1st National
Conference on Software Engineering
Washington D.C. 1975

/ROS 77/ Ross, D.T.
Structured Analysis: A Language
for Communicating Ideas
IEEE Transactions on Software
Engineering 3/1977

/SCHN 79/ Schnupp, P. u. Floyd, Ch.
Software, Programmentwicklung und
Projektorganisation
de Gruyter 1979

/SCHU 78/ Schulz, A.
Methoden des Software-Entwurfs und
strukturierte Programmierung
de Gruyter 1978

/SNE 80/ Sneed, H.
Software-Entwicklungsmethodik
Verl. Ges. R. Müller 1980

/SOF 81/ Programmentwicklungs-
Terminalsystem PET/1150
Firmenschrift der Fa. Softlab GmbH, München

/STE 81/ Stetter, F.
Softwaretechnologie
Bibliographisches Institut 1981
Reihe Informatik Bd. 33

/THA 78/ Thayer, T., Lipow, M. u. Nelson, E.
Software Reliability
North Holland 1978

Zusammenfassung

Ausgehend von den aktuellen Problemen bei der Produktion großer Softwaresysteme beschreibt die vorliegende Arbeit einen Lösungsvorschlag, der auf der mikrorechnerunterstützten halbautomatischen Projektverfolgung und -dokumentation mit Hilfe einer Beschreibungsbank beruht.

Durch kritische Analyse des heute allgemein anerkannten Phasenmodells der Softwareproduktion zeigen sich besondere Schwachstellen im Bereich der Schnittstellenkommunikation und der für die Wartung wichtigen Systemdokumentation.
Eine Betrachtung des Managementbereichs läßt die allgemeine Teamorganisation als ungeeignet erscheinen. Sie zeigt weiterhin, daß das 1970 von Baker und Mills vorgeschlagene Chief-Programmer-Team für Softwareprojekte durch den Einsatz von On-line-Systemen in einigen Punkten revidierungsbedürftig geworden ist. Speziell muß die für diese Struktur zentrale Stellung und Aufgabe des Projektsekretärs den heutigen Bedürfnissen angepaßt werden.
Unter Auswertung dieser Vorbetrachtungen basiert der Lösungsvorschlag der Arbeit auf einem datenbankähnlichen Beschreibungsbanksystem, das auf einem Mikrorechner installiert wurde. Dies in einer Master/Slave-Struktur aufgebaute System verwaltet unter der Regie des Projektsekretärs zentral allgemeine, managementbezogene und fachbezogene Daten des begleiteten Softwareprojekts. In einem Dialog in Menütechnik kann sich jeder Programmierer über das gesamte Projekt informieren; die Daten seines Arbeitsbereichs werden von ihm in einer halbformalen Form in die Beschreibungsbank eingetragen. Der Projektmanager hat jederzeit genauen Oberblick über den Produktionsverlauf.
Durch die Beschreibungsbank werden die verwalteten Daten nach verschiedenen Kriterien überprüft, speziell erfolgen Plausibilitätskontrollen der Eingaben, Syntaxüberprüfungen, Prüfungen auf Schnittstellenkompatibilität und Prüfungen des Ablaufverhaltens.
Die Weiterverarbeitung der Beschreibungsdaten erlaubt die Entlastung der Programmierer durch automatisch erzeugte standardisierte Dokumentation und Generierung von Quellprogrammrahmen.

Lebenslauf

1954	geboren am 6. Juni in Altena Westf.
1960 - 1964	Volksschule in Altena Westf.
1964 - 1972	Städt. Gymnasium mit mathem. naturwissenschaftlichem Zweig in Altena Westf.
1972	Industriepraktikum
1972 - 1978	Studium der Elektrotechnik an der Ruhr-Universität Bochum, Diplomarbeit am Lehrstuhl für Datenverarbeitung (Prof. Dr.-Ing. W. Weber)
1978 - 1982	Wissenschaftlicher Assistent am Lehrstuhl für Datenverarbeitung der Ruhr-Universität Bochum (Prof. Dr.-Ing. W. Weber)

Diese Arbeit entstand in den Jahren 1979 bis 1982 am Lehrstuhl für Datenverarbeitung der Ruhr-Universität Bochum.

Mein besonderer Dank gilt Herrn Prof. Dr. Weber für die gewährte wissenschaftliche Freiheit und für die stete Unterstützung in Rat und Tat.

Herrn Prof. Dr. Bender danke ich für die Obernahme des Korreferats, für die wertvollen Hinweise zur Struktur der Arbeit und nicht zuletzt auch für die selbstverständliche Oberwindung aller Probleme, deren Ursprung in der räumlichen Entfernung zwischen Karlsruhe und Bochum lag.

Schließlich möchte ich nicht vergessen, Frau Knippschild, Frau Burkhardt und Frau Dader für die Schreibarbeit am Manuskript dieser Arbeit und Frau Göbel für die Anfertigung der Zeichnungen herzlich zu danken.

Bochum, im Mai 1982 Ulrich George

D. GRÄBNER
DISSERTATIONS U. OFFSETDRUCKEREI

Gotenstraße 3
8602 Altendorf bei Bamberg
Telefon: 09545/8791